Títol: Simulador de xarxes de Petri

Volum: 1

Alumne: Guillem Català i Estapé

Director/Ponent: Pau Fonseca i Casas

Departament: EIO

Data: Desembre 2009

DADES DEL PROJECTE

Títol del Projecte: Simulador de xarxes de Petri

Nom de l'estudiant: Guillem Català i Estapé

Titulació: Enginyeria Tècnica Informàtica de Sistemes

Crèdits: 22.5

Director/Ponent: Pau Fonseca i Casas

Departament: Estadística i Investigació Operativa (EIO)

MEMBRES DEL TRIBUNAL *(nom i signatura)*

President:

Vocal:

Secretari:

QUALIFICACIÓ

Qualificació numèrica:

Qualificació descriptiva:

Data:

Índex

1. Introducció

1.1. Objectius

Aquest projecte es divideix en dos objectius principals.

El primer objectiu d'aquest projecte és trobar una representació en XML per a una xarxa de Petri Acolorida Temporal.

El segon és desenvolupar un editor amb entorn gràfic que sigui capaç d'interpretar aquest arxiu XML i el transformi en un model gràfic d'una xarxa de Petri per simular-ne el seu comportament.

2. Introducció a les xarxes de Petri

2.1. Què són les xarxes de Petri?

Una xarxa de Petri és un formalisme matemàtic usat per representar sistemes dinàmics basats en esdeveniments discrets. És una eina especialment útil per modelar i analitzar esdeveniments concurrents, asíncrons, distribuïts, paral·lels, no deterministes i/o estocàstics.

2.1.1. Breu Història

Les xarxes de Petri van ser definides per Carl Adam Petri (nascut el 12 de Juliol de 1926 a Leipzig). Matemàtic i informàtic, va desenvolupar les xarxes de Petri el 1962, com a part de la seva dissertació, "*Kommunikation mit Automaten*". Va treballar del 1959 al 1962 a la universitat de Bonn i va rebre el seu PhD el 1962 per la "Technische Universität Darmstadt".

2.1.2. Descripció Informal

Una xarxa de Petri es representa amb un graf dirigit, ponderat i bipartit. Té dos tipus de nodes: llocs i transicions.

Els llocs es connecten a les transicions i les transicions als llocs mitjançant arcs dirigits.

Els llocs es solen representar per cercles i les transicions per barres o rectangles.

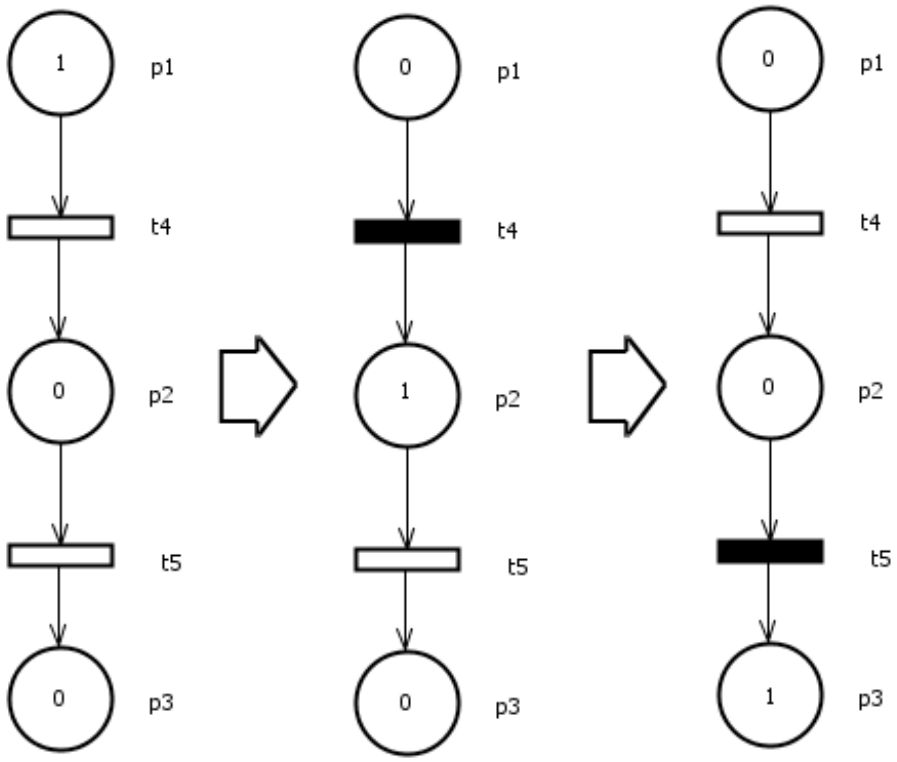
El comportament dinàmic d'una xarxa de Petri es controla per fitxes. Aquestes solen ser representades per petits punts negres dins dels cercles que representen els llocs. Varies fitxes poden residir dins d'un lloc al mateix instant.

Una xarxa de Petri s'executa quan una transició es dispara. Perquè una transició es dispari ha d'estar activada, i això ocorre quan hi ha una fitxa a cada lloc que

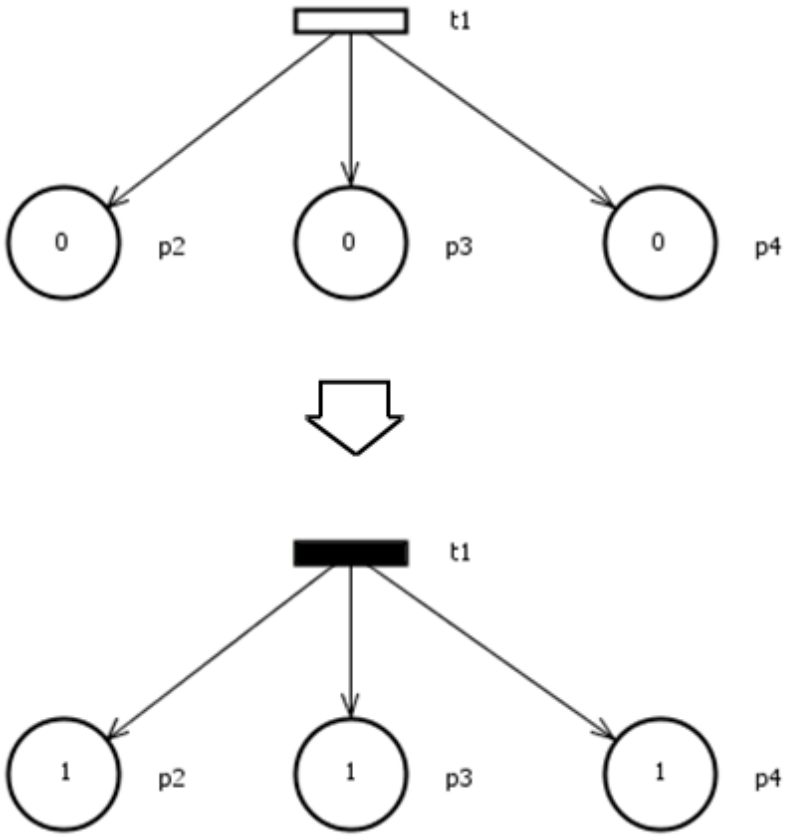
té un arc apuntant cap a la transició, o bé quan la transició no té cap arc d'entrada. Al disparar-se, la transició elimina les fitxes de cadascun dels llocs d'entrada, si n'hi ha, i crea una nova fitxa a cadascun dels llocs de sortida.

En general, una xarxa de Petri conté (segons les fitxes que hi hagi als llocs) un conjunt de transicions activades preparades per ser disparades. Si no hi ha transicions activades l'execució s'atura i la xarxa queda en punt mort. Si el conjunt té exactament una transició, aquesta es dispara. Tanmateix, si hi ha vàries transicions activades, llavors s'ha d'escollir quina serà la pròxima a disparar-se. La tria es pot fer aleatòriament, paral·lelament o controlada per un agent extern.

A continuació es mostra un exemple de xarxa de Petri clàssica on es veu l'evolució d'una fitxa en un procés seqüencial:



Un altre exemple d'execució d'un model de xarxa de Petri. Procés concurrent:



2.1.3. Definició Formal

Una xarxa de Petri Clàssica (PN) Z = (P,T,I,O,M_o) és una 5-tupla on:

P = {p_1,p_2,...,p_n}, n>0, es un conjunt finit de llocs representats per cercles;

T = {t_1,t_2,...t_s}, s>0, és un conjunt finit de transicions representades per rectangles on P∪T≠∅ i P∩T=∅;

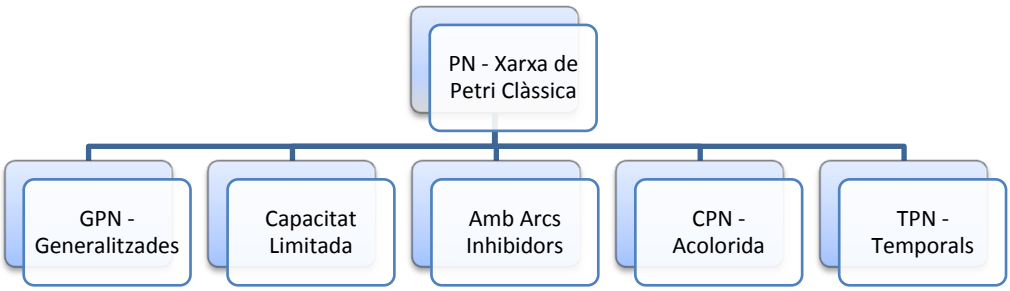
I:P×T→ N és una funció d'entrada que defineix un conjunt d'arcs dirigits de P a T on N ∈ ℕ;

O: P×T→ N, és una funció de sortida que defineix un conjunt d'arcs dirigits de T a P on N ∈ ℕ;

M_o: P→ N és la marca inicial: nombre inicial de fitxes a cada P.

2.2. Extensions de xarxes de Petri

A les xarxes de Petri clàssiques, se li han introduït extensions per optimitzar la potència del modelat o per aconseguir descripcions més concretes de l'eina. Algunes de les extensions són:



2.2.1. Generalitzades

GPN (Generalized Petri Net). Són aquelles en les que s'introdueix un pes als arcs. En les xarxes de Petri estàndard es permet que més d'un arc connecti a un lloc amb una transició o una transició amb un lloc. L'existència de varis arcs entre llocs i transicions complica la representació gràfica de la xarxa de Petri si la multiplicitat dels arcs és gran. Per simplificar el model de xarxes complicades, es permet una representació més compacte dels arcs múltiples que consisteix en connectar els diferents nodes amb un arc etiquetat. Aquesta etiqueta és un nombre natural i es denomina pes o valoració de l'arc. Per conveni, un arc no etiquetat té valoració d'una unitat.

2.2.2. Capacitat limitada

Són les xarxes de Petri en la que la capacitat de cada lloc per contenir fitxes està limitada per un valor màxim. Una transició estarà habilitada només si en tots els llocs d'entrada hi ha fitxes i si les fitxes generades al disparar-se no incompleix les restriccions de capacitat establertes.

2.2.3. Arcs inhibidors

Un arc inhibidor connecta un lloc amb una transició (el node inici sempre és un lloc) i es representa per una línia que acaba amb un cercle, en comptes d'una fletxa, cap a una transició. La condició de dispar d'una transició es generalitza dient que una transició està habilitada quan tots els llocs d'entrada normals contenen almenys tantes fitxes com arcs hi ha des de cada lloc a la transició, i no hi ha cap fitxa continguda en els llocs d'entrada units a les transicions per arcs inhibidors.

2.2.4. Acolorides

CPN (*Colored Petri Net*). És un tipus de xarxa de Petri d'alt nivell. El principal objectiu és reduir la mida del model, permetent marques individualitzades anomenades colors, que representen diferents processos o recursos en una mateixa subxarxa.

Inicialment, les marques de les xarxes acolorides estaven representades per colors simples que permetien diferenciar-les. Tanmateix, s'incrementa el potencial del modelatge considerablement si es permet la utilització de marques representades per estructures de dades complexes.

Per estendre encara més el model, també és possible executar operacions durant la seqüència de dispars de transicions.

Veiem a continuació una definició formal:

Una xarxa de Petri acolorida (CPN) $Z = (P,T,A,\Sigma,V,C,G,E,I_n)$ és una 8-tupla on:

P és un conjunt finit de llocs.

T és un conjunt finit de transicions.

A és un conjunt finit d'arcs on $P \cap T = P \cap A = T \cap A = \emptyset$;

Σ és un conjunt finit no buit de tipus, denominats colors.

V és un conjunt finit de variables on $Tipus(V) \in \Sigma$;

$C: P \rightarrow \Sigma$ és una funció de coloració.

G:T→exp és una funció de guarda, on exp és una expressió tal que $\forall t \in T$, Tipus(G(t)) = booleà i el Tipus(V(G(t))) $\in \Sigma$;

E:A→exp és una funció d'associació dels arcs on $\forall a \in A$, Tipus(a) = C i el Tipus(V(E(a))) $\in \Sigma$;

I_n:P→exp és una expressió d'inicialització on $\forall p \in P$, Tipus(I_n(p)) = C(p) i V(I_n(p)) = ϕ

2.2.5. Temporals

Per entendre com es representa una xarxa de Petri Temporal, primer s'ha de distingir clarament entre temps real i temps simulat. El temps real és el temps en el món físic on el simulador executa el model i on veiem què passa. El temps simulat és només una representació simbòlica del temps que podem afegir, opcionalment, a un model.

Per representar el temps en una xarxa de Petri hi ha vàries maneres. Una metodologia és associar una marca de temps a una fitxa, usualment el temps de creació, i afegir un rellotge de simulació. El rellotge de simulació és un comptador que marca quin és el temps de simulació.

Una fitxa no està disponible a menys que el temps del rellotge sigui més gran o igual que la seva marca de temps.

El rellotge s'incrementa en una unitat de temps sempre i quan hi hagi transicions activades però que no es puguin disparar degut a que alguna de les fitxes tingui una marca de temps superior a la que marqui el rellotge de simulació.

3. Representació de Xarxes de Petri en XML

Degut a la gran varietat de xarxes de Petri que hi ha, i al nombre d'eines per modelar-les, és difícil definir un document d'intercanvi XML que satisfaci totes les expectatives. Tot i que hi ha intents per estandarditzar les xarxes de Petri d'alt nivell com la ISO/IEC 15909 de pnml.org, encara no s'ha arribat a un acord definitiu. Es per això que PetriNetSim utilitza una versió adaptada d'aquesta proposta, pràcticament igual a la definida per l'aplicació Renew d'Olaf Kummer (un altre editor de xarxes de Petri que utilitza un subconjunt del llenguatge Java).

El motiu d'aquesta elecció està motivat perquè les especificacions de PNML per a xarxes de Petri d'alt nivell no preveuen l'ús de Java com a llenguatge per a l'avaluació i execució d'expressions, dificultant la integració del mateix codi Java amb la resta del model de la xarxa de Petri al format d'intercanvi XML.

També s'han estudiat altres propostes de formats XML com la que utilitza l'eina CPN Tools per guardar xarxes de Petri acolorides temporals. Aquesta eina utilitza el llenguatge CPN ML (construït a partir de l'estàndard del llenguatge funcional ML) per construir les declaracions i les inscripcions.

Un exemple de definició d'una variable en CPN ML seria del tipus:

```
<var id="ID9269271">
  <type><id>Y</id></type>
  <id>y</id>
  <layout>var y:Y;</layout>
</var>
```

Tenint cada variable un identificador propi i separant el tipus de la variable de la seva etiqueta.

A més les fitxes tenen la seva pròpia traducció a llenguatge CPN ML. Per exemple per afegir una nova fitxa a un lloc seria:

```
<text tool="CPN Tools" version="2.2.0">1`(1)++</text>
```

Per tant, s'observa que aquesta notació XML de CPN Tools està clarament adequada a l'ús del seu propi llenguatge de modelatge. Pel contrari, la proposta que presentarem a continuació mostra una estructura que, tot i que està pensada per Java, no exclouria a cap llenguatge de programació, ni tan sols al mateix ML.

3.1. El format de l'arxiu

A continuació s'explica com està format l'arxiu DTD per definir l'estructura dels arxius XML per representar les xarxes de Petri.

Cada arxiu XML que representi una xarxa de Petri començarà amb la definició de la mateixa xarxa amb l'etiqueta **net**. Seguidament hi haurà els llocs, les transicions, els arcs i les anotacions.

```
<!ELEMENT net (place*, transition*, arc*, annotation*)>
```

Els atributs de la xarxa són un identificador únic i el tipus de xarxa que és. En el nostre cas sempre serà CTPN.

```
<!ATTLIST net id ID #REQUIRED type CDATA #IMPLIED>
```

Els llocs i les transicions es defineixen de la mateixa forma. El tipus serà per defecte i implícitament "ordinary":

```
<!ELEMENT place (graphics?, annotation*)>
<!ATTLIST place id ID #REQUIRED type CDATA #IMPLIED>
<!ELEMENT transition (graphics?, annotation*)>
<!ATTLIST transition id ID #REQUIRED type CDATA #IMPLIED>
```

Per els arcs afegirem a més, el id del node origen i del node destí, i el tipus d'arc serà "ordinary" o "inhibitor".

```
<!ELEMENT arc (graphics?, annotation*)>
<!ATTLIST arc
  id ID #REQUIRED
  source IDREF #REQUIRED
  target IDREF #REQUIRED
  type CDATA #IMPLIED>
```

A la xarxa de Petri, als llocs, a les transicions i als arcs se'ls hi poden afegir anotacions, que consisteixen en informació textual que especifica el comportament de la xarxa o de l'element. Les anotacions també poden portar informació sobre la seva representació gràfica. Si el text no té significat, com en el cas d'una etiqueta, aquest es representa per un propi element <text>. Els tipus acceptats són: "name", "declaration", "initialmarking", "guard", "expression". A més, hem afegit els següents tipus: "import", "implement".

```
<!ELEMENT annotation (text, graphics?)>
<!ATTLIST annotation
  id ID #REQUIRED
  type CDATA #IMPLIED>
<!ELEMENT text (#PCDATA)>
```

L'element "graphics" determinarà les propietats gràfiques de cada element de la xarxa.

```
<!ELEMENT graphics (size?, textsize?, offset?,  fillcolor?,
pencolor?, textcolor?, point*, data*)>
<!ELEMENT size EMPTY>
<!ATTLIST size
  w CDATA #REQUIRED
  h CDATA #REQUIRED>
<!ELEMENT textsize EMPTY>
<!ATTLIST textsize
  size CDATA #REQUIRED>
<!ELEMENT offset EMPTY>
<!ATTLIST offset
  x CDATA #REQUIRED
  y CDATA #REQUIRED>
<!ELEMENT fillcolor (RGBcolor | transparent | background)>
<!ELEMENT pencolor (RGBcolor | transparent | background)>
<!ELEMENT textcolor (RGBcolor | transparent | background)>
<!ELEMENT RGBcolor EMPTY>
<!ATTLIST RGBcolor
  r CDATA #REQUIRED
  g CDATA #REQUIRED
  b CDATA #REQUIRED>
<!ELEMENT transparent EMPTY>
<!ELEMENT background EMPTY>
```

Els punts serviran per denotar els punts intermedis dels arcs.

```
<!ELEMENT point (x,y)>
<!ATTLIST point
  x CDATA #REQUIRED
  y CDATA #REQUIRED>
<!ELEMENT data (#PCDATA)>
<!ATTLIST data
  type CDATA #REQUIRED>
```

4. PetriNetSim

4.1. Introducció

PetriNetSim és un editor i simulador de xarxes de Petri que permet una perfecta integració de les xarxes de Petri Acolorides Temporals amb el llenguatge de programació d'alt nivell Java.

Tot i que existeixen altres eines per modelar, simular i fins i tot analitzar diferents tipus de xarxes de Petri, són escasses les aplicacions que accepten xarxes de Petri Acolorides Temporals. I encara més que siguin de codi obert, escrites en Java i que permetin introduir codi propi al model.

Després d'una intensa cerca, només eines com Renew http://www.renew.de/ o JFern https://sourceforge.net/projects/jfern/ (inacabada al moment d'escriure) podrien encaixar dins d'aquest grup. Una altra eina bastant completa és CPN TOOLS http://wiki.daimi.au.dk/cpntools/cpntools.wiki que no és de codi obert i utilitza una extensió del llenguatge ML per modelar les xarxes de Petri.

PetriNetSim és una aplicació pensada per a ús acadèmic. Específicament per a l'assignatura de Simulació i, per tant, s'ha considerat un seguit de mesures per tal de que el programa sigui fàcil d'estendre per afegir noves funcionalitats o modificar les ja existents, i també perquè sigui executat en un ampli ventall d'entorns. Concretament s'ha utilitzat Java com a llenguatge de programació degut a que és un llenguatge orientat a objectes que ajuda a fer programes estructurats i amb codi reutilitzable, i a més, pot ser executat en qualsevol sistema operatiu que disposi de la màquina virtual de Java instal·lada.

4.2. Requeriments

4.2.1. Interfície d'usuari i factors humans

PetriNetSim està orientat a enginyers o a estudiants d'informàtica o d'enginyeria. Tot i que l'aplicació està pensada per poder aprofitar el potencial de Java a l'hora de modelar i simular les xarxes de Petri, s'ha adequat el programa amb valors per defecte als arcs, transicions, etc per tal de que es puguin executar xarxes de Petri simples sense tenir gaires nocions del llenguatge. Tot i així, no hi ha més límits que els del propi llenguatge Java alhora de dissenyar i modelar xarxes més complexes.

4.2.2. Documentació

La documentació de l'aplicació consta de la documentació generada a partir de la utilitat Javadoc de Sun Microsystems, de l'explicació de com s'ha programat cadascuna de les àrees del software i d'un manual d'usuari complert amb explicacions i exemples per entendre les principals característiques i funcionalitats de l'aplicació.

4.2.3. Software

El programa s'ha desenvolupat en l'entorn NetBeans 6.5.1 utilitzant Java JDK 1.6.0.13.

La part gràfica s'ha desenvolupat utilitzant les llibreries gràfiques bàsiques de Swing.

Per avaluar les expressions de les declaracions, arcs i transicions s'ha utilitzat la API Janino que permet compilar en temps real el codi entrat per l'usuari.

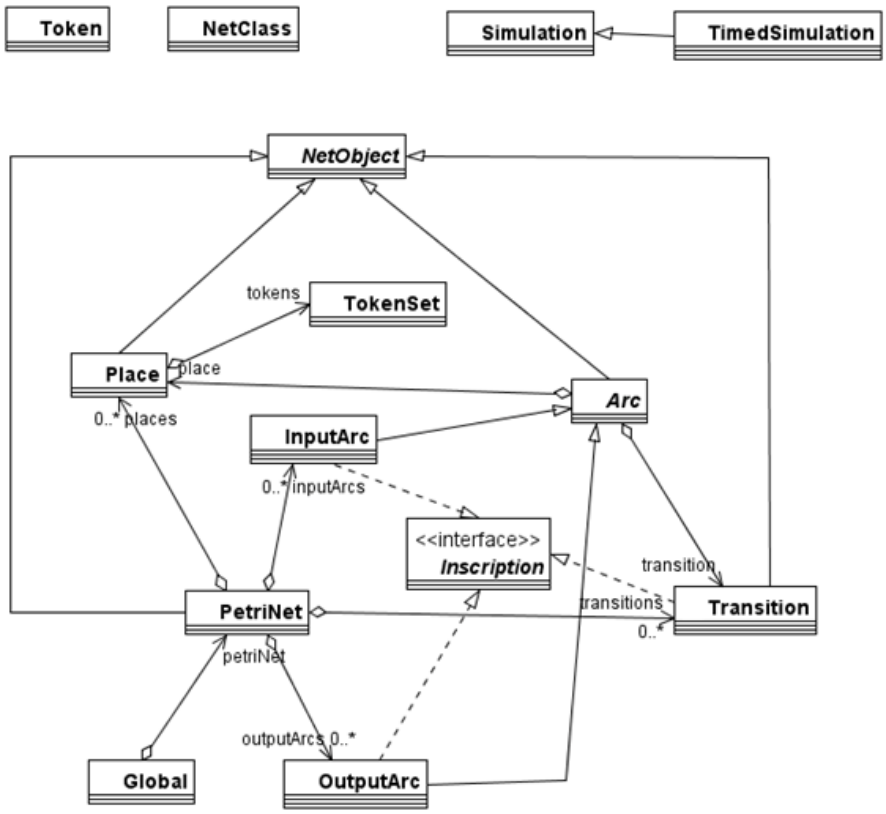
4.2.4. Tractament d'errors i condicions extremes

Degut a que les expressions o codi entrat per els usuaris és compilat en temps d'execució, és possible que amb freqüència hi hagi errors. Els usuaris són informats adequadament quan això ocorri.

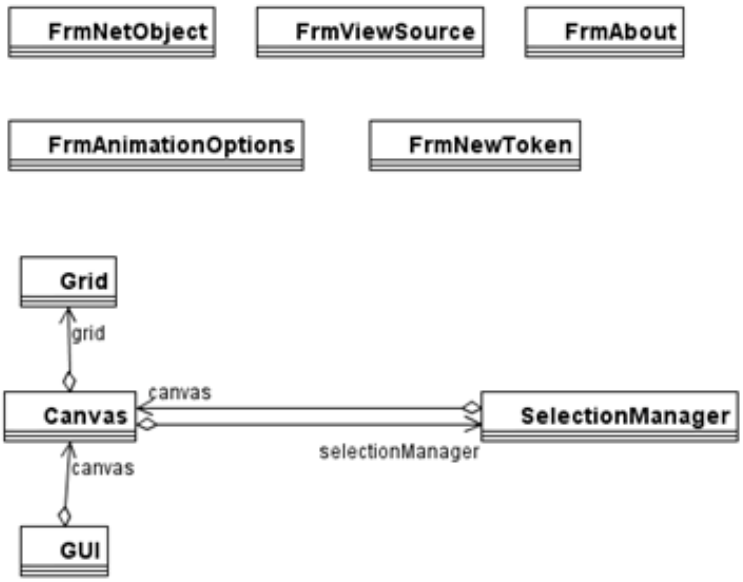
4.3. Especificació

4.3.1. Diagrama de classes

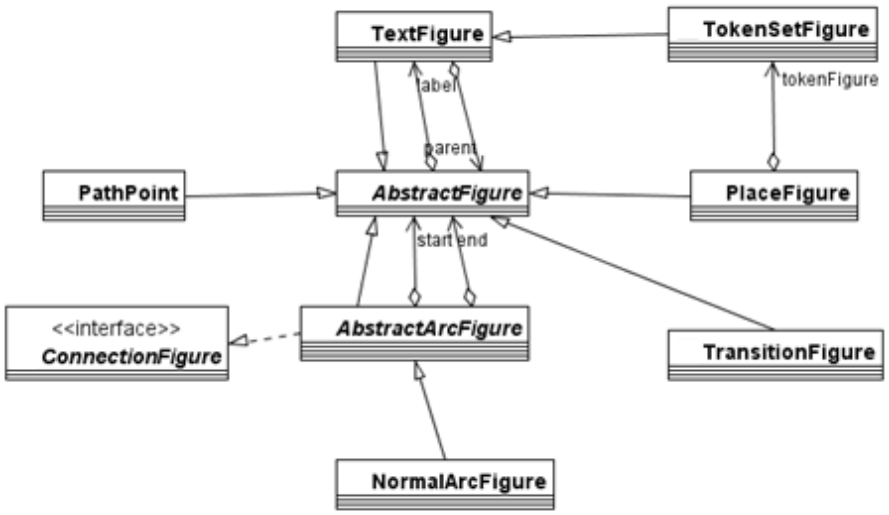
Capa Domini simplificada, sense mostrar atributs ni mètodes. Elements que representen la part lògica de la xarxa de Petri.



Capa Presentació. Aquesta capa està dividida entre la interfície gràfica de l'usuari i les figures que componen les xarxes de Petri.



Capa Presentació (Figures). Conjunt de classes per la representació de les figures que componen la xarxa de Petri.



Capa de Dades. Classe única que serveix per guardar i carregar les representacions de les xarxes de Petri en XML.

5. Explicació de la implementació

5.1. Compilar i executar el model de Xarxa de Petri

Per representar xarxes de Petri acolorides temporals, PetriNetSim permet utilitzar el mateix llenguatge de programació Java perquè es puguin crear i utilitzar models i patrons expressats en la forma de xarxes de Petri.

PetriNetSim permet programar les funcions de guarda de les transicions i dels arcs, les funcions d'execució dels arcs, la capacitat dels llocs i fins i tot importar altres classes al model o implementar interfícies tal i com qualsevol classe de Java pot fer. De fet, el model dissenyat des del programa, un cop és premut el botó d'execució, es converteix en una classe de Java que estén la classe PetriNet.

És per això que és necessari un sistema per compilar el codi definit de la xarxa en temps real per facilitar considerablement l'experiència de l'usuari. S'han estudiat diferents opcions i finalment s'ha escollit la API Janino per desenvolupar aquesta tasca.

5.1.1. Sobre Janino

Tot i que Java 6 incorpora una API per realitzar compilacions dinàmiques en temps real, presenta certs inconvenients. Un dels principals problemes que presenta és la necessitat d'afegir l'arxiu *tools.jar* al projecte, ja que JRE no ve per defecte amb un compilador. Això suposa uns 12 MB addicionals de pes per a l'aplicació. A més, l'API necessita molt de codi per desenvolupar una tasca aparentment senzilla com és passar-li una cadena de text, compilar-la i executar-la.

És per això que, finalment, s'ha utilitzat la API Janino (V. 2.5.15) per resoldre aquest problema. Aquesta eina és una llibreria JAR que al adjuntar-la al projecte ocupa menys de 500 KB. A més, és molt senzill compilar i executar una cadena de text sense necessitat de utilitzar massa codi. Veiem un exemple:

```
public void compilar(String javaSource){
        try {
            SimpleCompiler compiler = new SimpleCompiler();
            compiler.cook(new StringReader(javaSource));
            Class cl =
compiler.getClassLoader().loadClass("Nomdelaclase");
            NostreObjecte objecte = (NostreObjecte)
cl.newInstance();
        } catch (Exception e) {
            System.err.println(e.getMessage());
            e.printStackTrace();
        }
}
```

Cal dir també que actualment la API Janino només és compatible amb instruccions i funcionalitats de Java (JDK) 1.4 i que les funcionalitats més recents de Java 5 i Java 6 no es podran compilar.

5.1.2. Compilació del model

El codi mostrat a l'apartat anterior és només una mostra de com es realitza la compilació d'una classe. Abans però cal transformar el model gràfic de la xarxa de Petri a una cadena de text que tingui l'estructura d'una classe de Java. Això es fa a la classe NetClass, concretament amb la funció generateNetSource().

Cada vegada que es prem un dels botons de simulació del programa, a la classe GUI s'activa un esdeveniment que guarda la representació del model en una cadena de text abans de començar l'execució per, més tard, quan finalitzi la simulació, es pugui recuperar l'estat inicial del model. Concretament es guarda dins l'atribut javaSource.

5.2. Motor de simulació de la xarxa de Petri

La classe que controla la simulació és la classe TimedSimulation, extensió de la classe Simulation. Aquesta classe s'executa en un procés separat de la resta de l'aplicació. Quan s'executa el mètode run s'entra en un bucle del que no se'n sortirà fins que la xarxa de Petri entri en un estat de *deadlock*, o de finalització, on no hi hagi més transicions que es puguin disparar.

Dins d'aquest bucle s'executa el mètode isFinished de la classe TimedSimulation per comprovar l'estat de la xarxa de Petri. Aquesta funció obté una llista de transicions activades, i en cas que no n'hi hagi cap comprova si hi ha alguna fitxa amb marca de temps diferent de zero i inferior que el temps global del rellotge. Si es troba alguna fitxa que compleix aquest requisit, el rellotge passa a tenir la menor marca de temps trobada. Finalment es torna a comprovar una altra vegada si queden transicions activades. Si no n'hi ha cap s'entén que la xarxa de Petri ha entrat en un punt mort, és a dir, que no hi ha més transicions per ser disparades i la simulació finalitza.

S'ha separat en dues classes el motor de simulació TimedSimulation i Simulation per mostrar les diferències entre cadascuna. El programa però, executa directament la simulació temporal, entenent que una fitxa no és temporal si té una marca de temps igual a 0.

5.3. Expressions dels arcs i les transicions

Els arcs i les transicions tenen la possibilitat d'executar o avaluar expressions Java escrites pels usuaris. Les classes InputArc, OutputArc i Transition implementen la classe Inscription, que conté els mètodes evaluate (per avaluar expressions) i execute (per executar expressions) que es podran sobreescriure durant l'execució del programa.

A PetriNetSim només s'ha desenvolupat les funcions d'avaluació a les classes InputArc i Transition (conegudes com funcions de guarda) i les funcions d'execució a les classes InputArc i OutputArc.

Per cada un d'aquests mètodes que cada classe implementa hi ha un atribut que mostra la representació de l'expressió de forma textual: executeText i evaluateText. Aquests atributs són necessaris perquè l'usuari pugui recuperar la informació de les expressions un cop aquestes siguin compilades.

Per tal que els usuaris puguin configurar aquestes expressions amb l'editor, cal anar al formulari FrmNetObject.java que permet assignar els diferents paràmetres a les figures del model.

5.3.1. Dispar d'expressions d'execució

Aquestes expressions s'executen dins del programa durant la simulació del model de la xarxa de Petri i, concretament, al mètode fire() de la classe Transition.

Es recorreran tots els arcs d'entrada o de sortida connectats a la transició en qüestió i s'esborrarà (mètode removeTokens) o afegirà (mètode addToken) a cada lloc les fitxes generades per aquesta funció d'execució.

5.3.2. Comprovació d'expressions d'avaluació

Aquestes expressions s'avaluaran per poder determinar si una transició està activada per poder ser disparada. Al mètode enabled() de la classe Transition es recorren tots els arcs connectats a aquesta transició i es mira si l'expressió avalua a cert o a fals.

5.4. Capacitat dels llocs

Per representar llocs amb capacitat limitada (que només acceptin un número determinat de fitxes) s'ha afegit a la classe Place un atribut anomenat capacity. Per defecte val 0 i significa que no hi ha restricció de capacitat.

A la classe Transition, al mètode enabled() es comprova si el lloc en qüestió connectat a un arc de sortida compleix la restricció de capacitat.

Perquè l'usuari pugui modificar la capacitat d'un lloc del model, hi ha el formulari FrmNetObject que s'encarrega de gestionar-la.

Per a poder compilar el model, cal afegir la restricció de capacitat a la classe que es compilarà en temps real. Això es fa a la classe NetClass al mètode generateNetSource() dins la part d'inicialitzacions del constructor.

5.5. Generació de les figures

PetriNetSim permet pintar un conjunt de figures gràfiques que representen elements de la xarxa de Petri. Totes les figures estenen la classe AbstractFigure que conté les propietats bàsiques d'una figura. Excepte els arcs que estenen la classe AbstractArcFigure que alhora estén la classe AbstractFigure. Actualment PetriNetSim només permet utilitzar arcs normals, però fàcilment es podria donar suport a nous arcs, com arcs inhibidors. Per fer-ho caldria crear una classe semblant a la NormalArcFigure i sobreescriure el mètode generatePath() que s'encarrega de dibuixar un arc amb punta de fletxa normal.

Cada figura té un mètode per pintar el contingut drawFill() i un altre per pintar el contorn drawStroke(). Això permetrà definir i personalitzar la forma i colors per cada tipus de figura.

També cal dir que hi ha figures lligades a altres, per exemple, una etiqueta d'un lloc o transició es representa per la classe TextFigure i aquesta sempre ha de tenir constància d'un punt de referència, que és la posició de la figura que etiqueta. De forma que quan es mogui amb el ratolí la figura, també ho faci l'etiqueta. Aquest punt de referència és l'atribut offsetToParent que s'actualitza cada cop que es mou l'etiqueta amb la funció setPosition(). Un altre exemple és la figura TokenSetFigure que representa el comptador de fitxes que conté cada lloc.

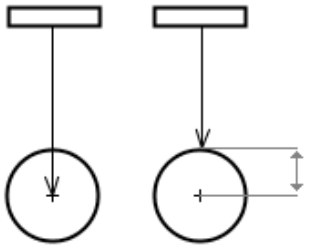
Com a última anotació sobre la generació de figures, hi ha una interfície anomenada ConnectionFigure que la classe AbstractArcFigure implementa, i que serveix per poder establir figures de connexió, és a dir, per representar quan un arc connecta a dues figures, una a cada extrem. En el nostre cas sempre serà la figura d'una transició i d'un lloc o al revés.

5.6. Generació d'arcs

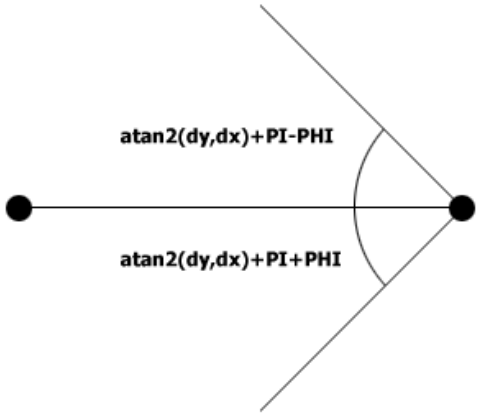
Els arcs que es dibuixen en el model estan representats dins la classe AbstractArcFigure. I concretament el mètode generatePath() s'encarrega de donar la forma de l'arc així com la punta de fletxa del mateix.

Un arc està compost per una llista de punts. I segons el nombre de punts definits tractarem la forma de l'arc d'una manera o una altra.

Normalment, un arc format tindrà com a mínim dos punts. Un de la figura inicial i un altre de la figura final. Aquests dos punts inicials no són exactament el centre d'aquestes figures, sinó que es calcula quin és el punt més llunyà de la figura sense sortir del seu contorn. Per exemple, si la figura és un lloc:



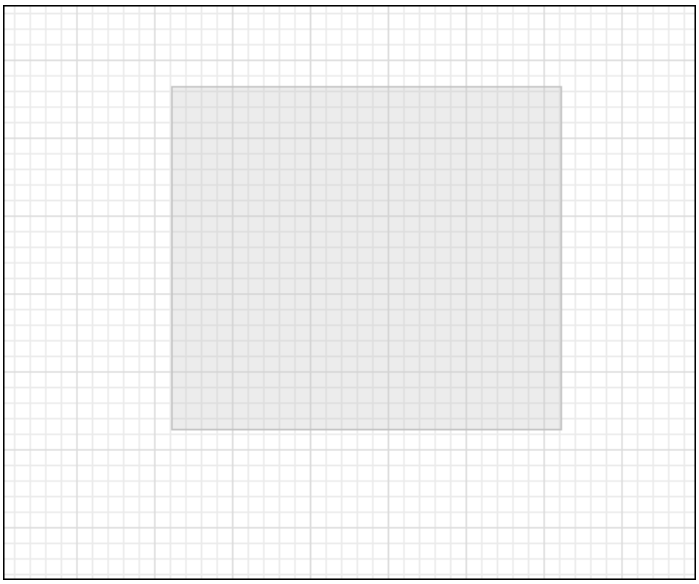
D'altra banda dins del mètode generatePath() també es dissenya la forma de la fletxa, per fer-ho tal tenir en compte l'orientació cap a on ha d'apuntar la fletxa. I això es fa a partir dels dos últims punts de l'arc. Aplicant la funció matemàtica atan2() a les respectives distàncies de cada eix de coordenades entre aquests dos últims punts. I després sumant-li PI en radians perquè s'orienti correctament cap al punt anterior.



PHI és l'angle en radians que mesurarà cada segment de la fletxa respecte la recta.

5.7. Selecció de figures del model

Dins l'àrea de dibuix de PetriNetSim s'ofereix la possibilitat de seleccionar les figures del model amb el ratolí prement en un punt de l'àrea de dibuix, movent el ratolí creant un rectangle de selecció i deixant de prémer. Totes les figures dins d'aquesta àrea rectangular passaran a estar seleccionades.



Per implementar aquest comportament, s'ha dissenyat la classe SelectionManager que representa una capa superposada a l'àrea de dibuix que és la classe Canvas. Aquesta superposició ocorre quan s'activa el mode de selecció. La capa en un estat inicial és transparent i ocupa la mateixa dimensió que l'àrea de dibuix. Per tant, el rectangle es pinta sobre aquesta capa.

La classe SelectionManager també ens permet afegir més funcionalitats, com per exemple que després de moure les figures, aquestes s'encaixin a la graella. Per això cal que l'atribut booleà snapToGrid estigui a cert. I la funció amb el mateix nom snapToGrid s'encarrega de col·locar les figures als punts de la graella més pròxims.

Les figures seleccionades es guarden en una estructura de dades de tipus HashMap dins la variable selectedFigures. Per pintar les figures seleccionades en l'àrea de dibuix es recorrerà aquesta estructura.

6. Documentació

6.1. Capa de Negoci

6.1.1. Global.java

Representa una classe que conté atributs per gestionar l'editor.

Atributs

petriNet:PetriNet. Conté la xarxa de Petri que està utilitzant l'aplicació.
mode:int. Representa l'estat en que es troba l'aplicació.
SELECTMODE:int. Quan l'aplicació està en mode selecció. Per indicar que es poden seleccionar i moure figures del disseny.
PLACEMODE:int. Quan s'afegeixen llocs al disseny.
TRANSITIONMODE:int. Quan s'afegeixen transicions al disseny.
NORMALARCMODE:int. Quan es poden afegir arcs al disseny.
SIMULATIONMODE:int. Quan la simulació del model està en curs.

6.1.2. Inscription.java

Serveix per donar l'habilitat als diferents components de la xarxa a avaluar expressions i executar-les.

evaluate:bool. Dóna l'habilitat d'avaluar-se.

execute:bool. Dóna l'habilitat d'executar-se.

getTokenSet:TokenSet. Dóna l'habilitat de poder obtenir un conjunt de fitxes segons el context.

6.1.3. NetObject.java

Representa un objecte de la xarxa de Petri.

Atributs

LATEST_ID: Long. Últim identificador assignat a un objecte de la xarxa.
id: String. Identificador assignat a aquest objecte.
label: String. Etiqueta assignada a aquest objecte.

Mètodes

getId():String. Retorna el identificador de l'objecte.
setId(String):void. Assigna un nou identificador a aquest objecte.
getLabel():String. Retorna l'etiqueta d'aquest objecte.
setLabel(String):void. Assigna una nova etiqueta a aquest objecte.

6.1.4. NetClass.java

Classe per transformar el model de la xarxa de Petri a una classe Java.

Atributs

netSource: StringBuffer. Conté el codi que representa la classe del model de la xarxa de Petri.
EOL:String. Representa la separació entre línies, indica el fi d'una línia.

Mètodes

NetClass. Constructor per crear una nova instància de la classe.
compile(String):void. Compila la cadena de text passada per paràmetre.
generateNetSource():String. Genera una cadena de text que conté el codi d'una classe que representa el model de la xarxa de Petri que s'està editant.
addSlashes(String):String. Retorna la mateixa cadena passada per paràmetre però afegint protecció a les cometes i salts de línia.
getNetSource():StringBuffer. Retorna el buffer que conté el codi de la classe.
setNetSource(StringBuffer):void. Afegeix un nou codi de classe.

6.1.5. PetriNet.java

Representa el model de la xarxa de Petri.

Atributs

places:Arraylist. Conté tots els llocs de la xarxa.
transitions:ArrayList. Conté totes les transicions de la xarxa.
inputArcs:ArrayList. Conté tots els arcs d'entrada de la xarxa.
outputArcs:ArrayList. Conté tots els arcs de sortida de la xarxa.
netElements:HashMap. Conté referències a tots els elements de la xarxa.
importText:String. Representa el text que servirà per importar altres classes i llibreries a la xarxa de Petri.
declarationText:String. Representa el text que contindrà els mètodes i atributs creats per l'usuari.
implementText:String. Representa el text que servirà perquè el model creat per l'usuari implementi altres classes.

Mètodes

PetriNet(). Constructor per a crear una nova xarxa de Petri.
isDead():bool. Retorna cert si hi ha transicions activades a la xarxa de Petri, fals altrament.
addPlace(Place):void. Afegeix un nou lloc a la llista de llocs.
addTransition(Transition):void. Afegeix una nova transició a la llista de transicions.
addInputArc(InputArc):void. Afegeix un nou arc d'entrada a la llista d'arcs d'entrada.
addOutputArc(OutputArc):void. Afegeix un nou arc de sortida a la llista d'arcs de sortida.
removePlace(Place):void. Elimina un lloc de la llista de llocs i tots els arcs d'entrada i de sortida que són connectats a ell.
removeTransition(Transition):void. Elimina una transició de la llista de transicions i tots els arcs d'entrada i de sortida que són connectats a ell.
removeInputArc(InputArc):void. Elimina un arc d'entrada de la llista d'arcs d'entrada.
removeOutputArc(OutputArc):void. Elimina un arc de sortida de la llista d'arcs de sortida.
removeInputArcs(String):void. Donat un identificador, elimina tots els

arcs d'entrada connectats a la figura que conté aquest identificador.
removeOutputARcs(String):void. Donat un identificador, elimina tots els arcs de sortida connectats a la figura que conté aquest identificador.
getPlaces():ArrayList. Retorna la llista dels llocs.
getTransitions():ArrayList. Retorna la llista de les transicions.
getInputArcs():ArrayList. Retorna la llista dels arcs d'entrada.
getOutputArcs():ArrayList. Retorna la llista dels arcs de sortida.
getNetElements():HashMap. Retorna el conjunt de tots els elements de la xarxa.

6.1.6. Arc.java

Representa un arc. Un arc està composat per un lloc i una transició.

Atributs

place:Place. Lloc connectat a aquest arc.
transition:Transition: Transició connectada a aquest arc.

Mètodes

getPlace():Place. Retorna el lloc connectat a aquest arc.
setPlace(Place):void. Assigna un nou lloc a l'arc.
getTransition():Transition. Retorna la transició connectada a aquest arc.
setTransition(Transition):void. Assigna una nova transició a l'arc.

6.1.7. InputArc.java

Representa un arc d'entrada, és a dir, un arc que apunta cap a una transició.

Atributs

evaluateText:String. Representa el text de l'expressió que l'arc avaluarà.
executeText:String. Representa el text de l'expressió que l'arc executarà.

Mètodes

InputArc(Place,Transition). Constructor per a crear un arc d'entrada.
InputArc(String,Place,Transition). Constructor per a crear un arc d'entrada donat un identificador.
evaluate():boolean. Mètode que s'avalua per comprovar si un arc està activat.
execute():TokenSet. Mètode que s'executa quan es dispara una transició.
getTokenSet():TokenSet. Retorna les fitxes que conté el lloc connectat a aquest arc.
removeTimedToken(TokenSet):Token. Donat un conjunt de fitxes retorna una fitxa amb marca de temps menor o igual al rellotge de simulació.
getEvaluateText():String. Retorna l'expressió d'avaluació de l'arc.
setEvaluateText(String):void. Assigna una nova expressió d'avaluació.
getExecuteText():String. Retorna l'expressió d'execució de l'arc.
setExecuteText(String):void. Assigna una nova expressió d'execució.

6.1.8. OutputArc.java

Representa un arc de sortida, és a dir, un arc que apunta cap a un lloc.

Atributs

executeText:String. Representa el text de l'expressió que l'arc executarà.

Mètodes

OutputArc(Place,Transition). Constructor per a crear un arc de sortida.
OutputArc(String,Place,Transition,String). Constructor per a crear un arc de sortida donat un identificador, un lloc, una transició i el text d'una expressió d'execució.
evaluate():boolean. No implementat.
execute():TokenSet. Mètode que s'executa quan es dispara una transició.
getTokenSet():TokenSet. Retorna les fitxes que conté el lloc connectat a aquest arc.
getExecuteText():String. Retorna l'expressió d'execució de l'arc.
setExecuteText(String):void. Assigna una nova expressió d'execució.

6.1.9. TokenSet.java

Representa un contenidor de fitxes.

Atributs

tokenList:ArrayList. Llista de fitxes que representa un conjunt.

Mètodes

TokenSet(Object). Constructor per afegir un nou TokenSet. L'objecte pot ser una instància de Token, una instància de TokenSet o un objecte de Java.
TokenSet(Object,long). Constructor per afegir un nou TokenSet amb una fitxa amb marca de temps.
TokenSet(Object,String). Constructor per afegir un nou objecte i una expressió de creació inicial de la fitxa.
TokenSet(Object,long,String). Constructor per afegir un nou objecte, una expressió de creació inicial de la fitxa i una marca de temps a aquesta fitxa.
iterator():Iterator. Retorna l'iterador per recórrer aquest conjunt.
size():int. Retorna el número d'elements que conté aquest conjunt.
add(Object):boolean. Afegeix un nou element al conjunt.
addAll(Collection):boolean. Afegeix un conjunt existent al conjunt actual.
get(int):Token. Retorna una fitxa del conjunt donat un índex.
remove(Object):boolean. Elimina un element del conjunt.
clear():void. Buida el conjunt d'elements.
removeAll(Collection):boolean. Elimina els elements del conjunt que estiguin continguts al conjunt passat per paràmetre.
containsTime(Long):boolean. Retorna cert si existeix almenys una fitxa amb marca de temps que tingui un temps menor al passat per paràmetre. Fals altrament.
incrementTime(Long):void. Incrementa les fitxes amb marca de temps amb la quantitat fixa passada per paràmetre.

6.1.10. Token.java

Representa una fitxa.

Atributs

initialMarkingExpression:String. Cadena de text que representa a l'expressió de creació de la fitxa.
object:Object. Objecte que conté la fitxa.
timestamp:Long. Marca de temps.

Mètodes

Tokent(Object). Constructor que crea una nova fitxa sense marca de temps.
Token(Object,long). Constructor usat per crear fitxes amb marca de temps.
Token(Object,long, initialMarkingExpression). Constructor usat per crear fitxes amb marca de temps i expressió inicial de creació de la fitxa.
equals(Object):boolean. Comprova si l'objecte passat per paràmetre és igual a l'objecte de la fitxa.
toString():String. Retorna una representació en forma cadena de text de la fitxa.
getObject():Object. Retorna l'objecte de la fitxa.
setObject(Object):void. Assigna un nou objecte a la fitxa.
getTimestamp():long. Retorna la marca de temps associada a la fitxa.
setTimestamp(long):void. Assigna una nova marca de temps a la fitxa.
getInitialMarkingExpression():String. Retorna l'expressió de creació de la fitxa.
setInitialMarkingExpression(String):void. Assigna una nova expressió de creació de la fitxa.

6.2. Capa de Dades

6.2.1. FileManager.java

Representa la classe per guardar i carregar models de xarxes de Petri a l'editor.

Atributs

dom:Document. Representa el document on es carregarà o guardarà tot el model.
pnml:Element. Element de suport per a la creació de nodes xml.

Mètodes

loadFile(File):HashMap. Donat un arxiu passat per paràmetre, carrega el model que representa al programa i retorna un HashMap amb les figures gràfiques.
generateXML(HashMap, File). Donades les figures gràfiques i un arxiu, guarda el model de la xarxa de Petri en format XML.
saveLabel(TextFigure,String):Element. Retorna un node XML amb nom d'etiqueta que es passa per paràmetre.
createGraphic(AbstractFigure):Element. Retorna un node XML que representa la figura gràfica passada com a paràmetre.
savePNG(BufferedImage,File):void. Guarda la imatge passada per paràmetre a l'arxiu passat per paràmetre.

6.3. Capa de Presentació

6.3.1. AbstractFigure.java

Representa una figura gràfica bàsica.

Atributs

position:Point2D. Coordenades de la figura.
offset:Point2D. Punt de referència.
label:TextFigure. Representa la TextFigure associada a aquesta figura.
fillColor:Color. Color amb que es pintarà l'interior de la figura.
strokeColor:Color. Color amb que es pintarà el contorn de la figura.
selectedColor:Color. Color amb que es pintarà l'interior de la figura quan aquesta estigui seleccionada.
hihglightedColor:Color. Color amb que es pintarà l'interior de la figura quan aquesta estigui resaltada.
selected:boolean. Cert si la figura està seleccionada. Fals altrament.
highlighted:boolean. Cert si la figura està resaltada. Fals altrament.

Mètodes

contains(Point2D):boolean. Determina si la figura conté les coordenades passades per paràmetre.
draw(Graphics2D):void. Serveix per pintar el conjunt de la figura.
drawFill(Graphics2D):void. Serveix per pintar l'interior de la figura.
drawStroke(Graphics2D):void. Serveix per pintar el contorn de la figura.
getElementId():String. Retorna el identificador de la figura.
setElementId(String). Assigna un nou identificador a la figura.
getBounds():RectangularShape. Retorna la forma de la figura que conté les dimensions.
getLabel():TextFigure. Retorna la figura TextFigure associada a aquesta figura.
setLabel(TextFigure):void. Assigna una nova TextFigure a aquesta figura.
getPosition():Point2D. Retorna la posició d'aquesta figura.
setPosition(Point2D):void. Assigna una nova posició a la figura.
isSelected():boolean. Retorna cert si la figura està seleccionada. Fals altrament.

setSelected(boolean):void. Assigna el valor passat per paràmetre.
isHighlighted():boolean. Retorna cert si la figura està ressaltada. Fals altrament.
setHighlighted(boolean):void. Assigna el valor passat per paràmetre.
getOffset():Point2D. Retorna les coordenades de referència.
setOffset(Point2D):void. Assigna unes noves coordenades de referència.

6.3.2. AbstractArcFigure.java / NormalArcFigure.java

Representa una figura d'arc gràfic bàsic. S'ha fet una abstracció i s'ha separat en dues classes per poder afegir diferents tipus d'arcs en futures implementacions.

Atributs

start:AbstractFigure. Figura inicial connectada a aquesta figura.
end:AbstractFigure. Figura final connectada a aquesta figura.
arcId:String. Identificador de l'arc.
path:GeneralPath. Representa la forma de l'arc.
pathPoints:LinkedList. Representa el conjunt de punts intermedis que formen l'arc. No es considera punt intermedi ni el punt inicial de la figura ni el final.
BARB:int. Tamany de la punta de la fletxa.
PHI:double. Angle de la punta de la fletxa.
selectedPoints:HashMap. Conjunt de punts intermedis seleccionats.

Mètodes

addPoint(Point2D):void. Crea un nou punt intermedi.
removePoint(PathPoint):void. Elimina el punt intermedi passat per paràmetre.
setConnectionStart(AbstractFigure):void. Assigna una nova figura com a figura inicial.
getStartConnector():AbstractFigure. Retorna la figura d'inici connectada a l'arc.
setConnectionEnd(AbstractFigure):void. Assigna una nova figura com a figura final.
getEndConnector():AbstractFigure. Retorna la figura final connectada a l'arc.
setLabel():void. Crea una nova TextFigura assignada a aquesta figura.
setSelectedPoint(Point2D):void. Assigna un nou punt al conjunt de punts seleccionats.
removeSelectedPoints():void. Elimina tots els punts seleccionats.
ContainsPoint(Point2D):Point2D. Comprova si una coordenada donada és a prop d'algun dels punts intermedis de l'arc.

getIntersectingPoint(double, Rectangle2D). Retorna el punt d'intersecció entre una línia i un rectangle.
generatePath():void. Crea la figura que representa l'arc.
round(Double):int. Usat per arrodonir.
operation():float. Operació per calcular la distància entre el centre de la figura i el seu marge.
getPoint(Point2D,double). Distància per obtenir el punt final de la fletxa.
getLine(Point2D,double). Retorna la línia de la punta de la fletxa.
getPoints():LinkedList. Retorna els punts intermedis de l'arc.
setPoints(LinkedList):void. Assigna un nou conjunt de punts intermedis a l'arc.

6.3.3. ConnectionFigure.java

Dóna l'habilitat a un objecte de poder tenir altres figures connectades a ell.

Mètodes

setConnectionStart(AbstractFigure):void. Permet afegir una figura d'inici.

getStartConnector():AbstractFigure. Permet obtenir la figura d'inici.

setConnectorEnd(AbstractFigure):void. Permet afegir una figura final.

getEndConnector():AbstractFigure. Permet obtenir la figura final.

6.3.4. PathPoint.java

Figura gràfica que representa el punt intermedi d'un arc.

Atributs

POINTSIZE:int. Dimensió del punt intermedi.
rectangle:Rectangle2D. Rectangle que representa el contorn del punt intermedi que es dibuixarà.
id:String. Identificador del punt intermedi.

Mètodes

PathPoint(Point2D,String). Constructor que crea un nou punt intermedi.

6.3.5. PlaceFigure.java

Figura gràfica que representa un lloc.

Atributs

placeId:String. Identificador de la figura.
ellipse:Ellipse2D. Forma que representa el lloc.
DIAMETER:int. Diàmetre de la figura.

Mètodes

PlaceFigure(String,Point2D). Constructor per crear una nova figura donat un identificador i una coordenades.
generateEllipse():Ellipse2D. Retorna la forma del lloc que és una el·lipse.

6.3.6. TransitionFigure.java

Figura gràfica que representa una transició.

Atributs

WIDTH:int. Ample de la transició.
HEIGHT:int. Alçada de la transició.
rectangle:Rectangle2D. Forma que representa la transició.
transitionId:String. Identificador de la figura.

Mètodes

TransitionFigure(String,Point2D). Constructor per crear una nova figura donat un identificador i una coordenades.

6.3.7. TextFigure.java

Figura gràfica que representa una etiqueta d'una altra figura.

Atributs

offsetToParent:Point2D. Coordenada de separació entre aquesta figura i el seu pare.
parent:AbstractFigure. Figura de la qual l'etiqueta n'està associada.
rectangle:Rectangle2D. Forma que representa l'etiqueta.

Mètodes

getText():String. Retorna una cadena que conté l'etiqueta de la figura juntament amb el l'identificador de la figura a la que està associada.
getTextLabel():String. Retorna l'etiqueta de la figura a la que està associada.
setRelativePosition(Point2D). Assigna una nova posició de l'etiqueta en relació a la coordenada de la figura a la que està associada.
getOffsetToParent():Point2D. Retorna la coordenada de separació entre aquesta figura i el seu pare.
setOffsetToParent(Point2D). Assigna una nova coordenada de separació.

6.3.8. TokenSetFigure.java

Figura gràfica que representa el número de fitxes que conté un lloc.

Atributs

placeId:String. Identificador del lloc a que està associada aquesta figura.
ellipse:Ellipse2D. Forma que representa aquesta figura.
offsetToParent:Point2D. Coordenada a que es troba la figura a que està associada.
DIAMETER:int. Diàmetre de l'el·lipse que representa aquesta figura.

Mètodes

generateEllipse():Ellipse2D. Crea l'el·lipse que representa aquesta figura.

6.3.9. SelectionManager.java

Classe que s'encarrega de gestionar la selecció d'elements gràfics.

Atributs

selectionStartPoint:Point2D. Coordinades inicials del rectangle de selecció.
selectionEndPoint: Point2D. Coordinades finals del rectangle de selecció.
selectionRectangle:Rectangle2D. Rectangle usat per seleccionar múltiples elements.
selectionEnabled:boolean. Cert per habilitar la selecció de figures. Fals altrament.
selectedFigures:HashMap. Conjunt de figures seleccionades.
fillColor:Color. Color per pintar l'interior del rectangle.
strokeColor:Color. Color per pintar el contorn del rectangle.
canvas:Canvas. Component pare que conté aquest component SelectionManager.

Mètodes

SelectionManager(Canvas). Constructor per crear una nova instància.
updateBounds():void. Actualitza les dimensions d'aquest component.
paintComponent(Graphics):void. Pinta els elements seleccionats i el rectangle de selecció.
calculateRectangleBounds(Point2D):void. Calcula les dimensions del rectangle de selecció.
addSelectedFigure(AbstractFigure):void. Assigna una figura com a seleccionada i l'afegeix al conjunt de figures seleccionades.
addSelectedElements(HashMap):void. Donat un conjunt de figures, determina les que estan contingudes dins del rectangle de selecció
removeSelectedFigures():void. Desmarca les figures seleccionades.
updateOffsets(Point2D):void. Actualitza les distàncies de referència de totes les figures seleccionades respecte el punt donat.
getSelectionStartPoint():Point2D. Retorna el punt d'inici del rectangle de selecció.
setSelectionStartPoint(Point2D):void. Assigna una nova coordenada d'inici del rectangle de selecció.
getSelectionEndPoint():Point2D. Retorna la coordenada final del rectangle

de selecció.
setSelectionEndPoint(Point2D). Assigna una nova coordenada final del rectangle de selecció.
getSelectionRectangle():Rectangle2D. Retorna la forma que representa el rectangle de selecció.
setSelectionRectangle(Rectangle2D):void. Assigna un nova forma al rectangle de selecció.
setSelectionEnabled(boolean):void. Si el paràmetre és cert, habilita la selecció. Si és fals la deshabilita.
snapToGrid(Point2D). Donat una coordenada, col·loca totes les figures seleccionades encaixades a la graella en referència a aquesta coordenada.
getSelectedElements():HashMap. Retorna el conjunt de les figures seleccionades.
setSelectedElements():void. Assigna un nou conjunt de figures seleccionades.
mousePressed(MouseEvent):void. Controla l'event de prémer el ratolí. Selecciona una figura, habilita el botó dret per accedir als detalls de cada figura o assigna un punt inicial del rectangle de selecció.
mouseReleased(MouseEvent):void. Controla l'event de deixar de prémer el botó del ratolí. Selecciona els elements dins del rectangle de selecció.
mouseDragged(MouseEvent):void. Controla l'event de arrastrar el ratolí amb un botó del mateix premut. Arrastra les figures seleccionades si n'hi ha o augmenta les dimensions del rectangle de selecció.
keyPressed(KeyEvent):void. Elimina les figures seleccionades quan es prem la tecla 'supr'.

6.3.10. Grid.java

Representa la graella sobre la qual es mostraran els elements gràfics.

Atributs

width:int. Amplada de la graella.
height:int. Alçada de la graella.
cellSize:int. Dimensions de la graella.
strongColor:Color. Color dels requadres grans.
weakColor:Color. Color dels requadres petits.
backgroundGrid:GeneralPath. Forma dels requadres grans.
foregroundGrid:GeneralPath. Forma dels requadres petits.

Mètodes

Grid(int,int). Constructor per crear una nova graella donades una amplada i una alçada.
generateGrid(int):GeneralPath. Crea la forma de la graella amb un nombre predefinit de columnes petites dins de cada columna gran.
drawGrid(Graphics2D):void. Pinta la graella.

6.3.11. FrmAnimationOptions.java

Representa la pantalla de configuració de l'animació.

Mètodes

FrmAnimationOptions(Frame,boolean). Constructor per crear una nova finestra d'opcions d'animació.

jButton1ActionPerformed(ActionEvent):void. Mètode per guardar els canvis realitzats.

6.3.12. GUI.java

Representa l'editor la interfície gràfica principal de l'editor.

Atributs

simulator:TimedSimulation. Referència al procés de simulació.
defaultPath:String. Directori principal a on es mirarà al intentar obrir un nou model.
javaSource:String. La classe Java que representa el model en format de text.
buttonGroup1:ArrayList. Subconjunt de botons de la barra d'eines.

Mètodes

GUI(). Constructor per crear la nova interfície quan s'obra l'aplicació.
setEnabledButtons(JButton,ArrayList,boolean):void. Activa o desactiva els botons de la llista del segon paràmetre excepte el botó del primer paràmetre.
enableControlButtons(boolean):void. Activa o desactiva el botó d'stop.
deactivateButtons(JToggleButton):void. Deselecciona els botons de tipus Toggle que estàn activats excepte el passat per paràmetre.
continuousSimulation(ActionEvent):void. Inicia el procés de simulació tot guardant el codi que representa el model per restaurar-lo més endavant.
steppedSimulation(ActionEvent):void. Inicia o continua el procés de simulació per passos (dispar d'una transició cada pas).
stopSimulation(ActionEvent):void. Interromp el procés de simulació restaurant el model a l'estat inicial.
newNet():void. Crea una nova xarxa de Petri tot inicialitzant els objectes necessaris.
openFile():void. Obra un nou arxiu que representi un model de xarxa de Petri.
saveFile():void. Guarda el model actual de l'editor en un arxiu.
setSelectionMode():void. Activa o desactiva la propietat de seleccionar figures del model gràfic.
getCanvas():Canvas. Retorna el component de dibuix on es representen totes les figures de la xarxa.
getTxTClock():JTextField. Retorna el component on es representa el valor del rellotge de simulació Global.

6.3.1. Canvas.java

Representa l'àrea de dibuix sobre la qual es dibuixaran les figures del model.

Atributs

selectionManager:SelectionManager. Capa de selecció.
figures:HashMap. Conté referències a les figures gràfiques del model.
arcFigure:AbstractArcFigure. Figura d'un arc que servirà per dibuixar un arc temporal.
grid:Grid. Graella de fons sobre la qual es dibuixa el model.
enableGrid:boolean. Cert si es vol fer visible la graella. Fals altrament.

Mètodes

Canvas(). Constructor per crear una nova instància de la classe.
paintComponent(Graphics):void. Mètode que s'encarrega de pintar l'àrea de dibuix i tots els seus components així com les figures del model de la xarxa de Petri.
drawCanvas():BufferedImage. Mètode que retorna una imatge del model. S'usarà per exportar una imatge en un arxiu.
showForm(NetObject):void. Mostra el formulari d'edició d'objectes de la xarxa de Petri.
addFigure(int,Point2D):void. Afegeix una figura al disseny i l'element que representa al model.
removeFigure(AbstractFigure):void. Elimina la figura de l'àrea de dibuix així com del model.
removeArcFigures(String):void. Elimina totes les figures que componen un arc.
removePathPoints(AbstractArcFigure):void. Elimina tots els punts entremitjos de l'arc passat per paràmetre.
selectFigure(Point2D):AbstractFigure. Donat un punt de l'àrea de dibuix, retorna la figura que el conté.
getFigureKey(AbstractFigure):String. Retorna l'identificador d'una figura.
findDuplicatedArc(AbstractArcFigure):boolean. Retorna cert si ja existeix un arc igual que el passat per paràmetre. Retorna fals altrament.
addArc(Point2D):void. Afegeix un nou arc al model, així com la figura que el representa a l'àrea de disseny.

highlightPlaces(ArrayList,String,boolean,boolean):void. Il·lumina els llocs passats per paràmetre connectats a la l'identificador de la transició també passada per paràmetre. El primer booleà serveix per determinar si s'i·lumina o s'apaga la figura i el segon per si cal afegir un temps d'espera o no.
highlightArcs(ArrayList,String,boolean,boolean):void. Igual que l'anterior però per figures d'arcs.
highlightTransition(String,boolean,boolean):void. Igual que l'anterior però per la figura d'una transició.
sleepRepaint():void. Avisa perquè s'actualitzi l'àrea de dibuix i afegeix un temps d'espera determinat.
snapPointToGrid(Point2D):Point2D. Situa un punt passat per paràmetre encaixat a la graella.

7. Annex A: Manual d'usuari

7.1. Requeriments

Cal tenir instal·lada la versió de Java JRE 6.0 o superior per poder executar PetriNetSim.

És necessari que es puguin executar les instruccions Java des de qualsevol ruta del sistema, per aconseguir-ho cal configurar correctament les variables d'entorn. En les versions de Java que s'han provat la instal·lació ja ho configurava, tot i així, si sorgeixen errors al executar el programa, és possible que aquesta en sigui la causa més probable.

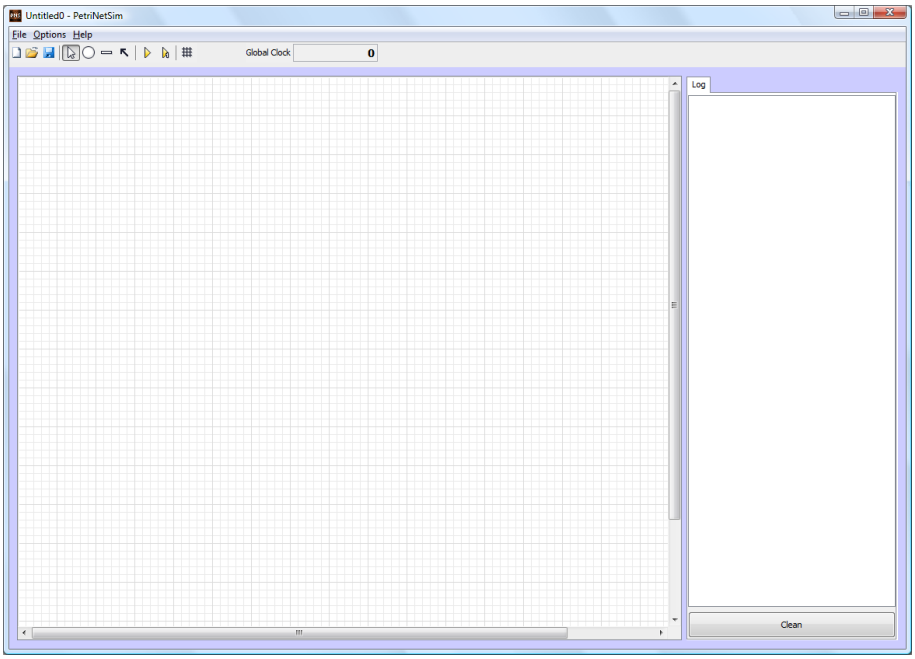
Es requereix sistema operatiu amb entorn gràfic.

7.2. Execució

PetriNetSim no requereix instal·lació. Es pot executar a partir de l'arxiu start.bat.

7.3. Pantalla Inicial

Quan s'obra l'aplicació ens trobem amb una pantalla com aquesta.



Està principalment dividida en 3 parts:

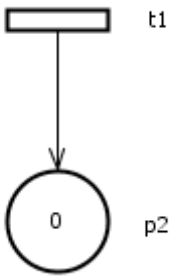
- La barra d'eines i els botons. Per gestionar les accions de l'aplicació.
- El registre. On quedaran reflectides les funcions de dispar de les transicions.
- L'àrea de dibuix. És on es dibuixarà el model gràfic de la xarxa de Petri.

7.4. Creació de la primera xarxa de Petri

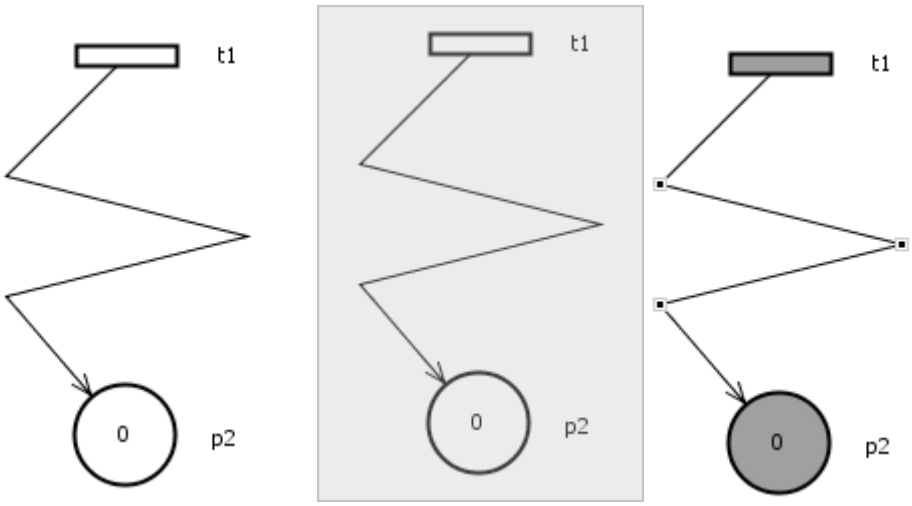
Per defecte, al obrir l'aplicació, ja ens trobem en mode SELECCIÓ (). Cal destacar que només podrem seleccionar i moure elements quan aquest botó de la barra d'eines estigui activat.

Per crear una nova xarxa de Petri podem prémer el botó () o prémer ctrl+N. Per defecte també es crea una xarxa de Petri en blanc al obrir l'aplicació. En aquest primer exemple crearem una xarxa de Petri simple amb una transició i un lloc connectats per un arc.

Primer premerem el botó de la barra d'eines () llavors premerem sobre l'àrea de dibuix on es vulgui col·locar la transició. Després seleccionarem el botó () i col·locarem de la mateixa forma el lloc. Finalment crearem un arc de sortida, és a dir, un arc que vagi des de la transició com a posició inicial fins al lloc com a posició final. Seleccionem el botó () fem un clic sobre la transició i després un altre sobre el lloc. La xarxa resultant ha de ser com la de la figura.



També és possible fer arcs amb punts intermedis. Aquests es fan prement a espais en blanc allà on es vulguin afegir els punts mentre es crea l'arc amb el botó (). Aquests punts ens permetran fer xarxes més vistoses i ordenades quan els models siguin més complexes. Igualment, amb l'eina de selecció, podrem seleccionar tots els punts i figures del model i moure'ls junts o per separat a una nova posició.

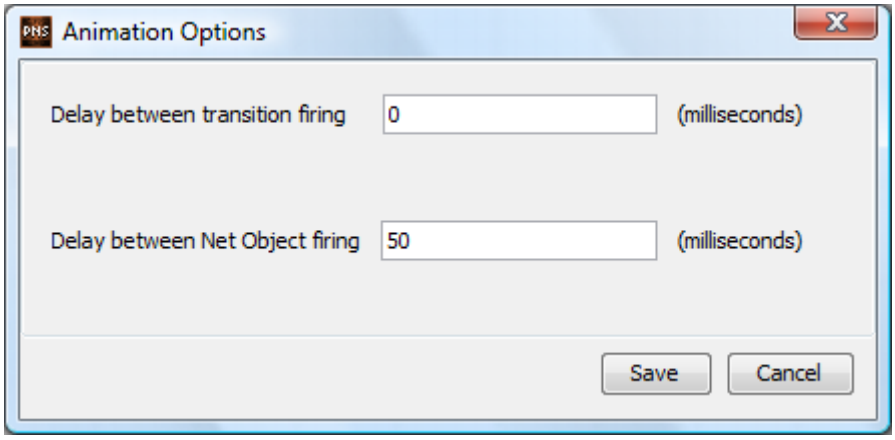


Es poden eliminar elements seleccionats prement la tecla ‘supr’.

7.5. Simulació del primer model

Un cop dissenyat el primer model, ja es pot executar prement el botó ( ) per simular fins a un punt de DEADLOCK o ( ) per simular el dispar d'una única transició.

Hi ha un parell d'opcions per configurar com es veurà l'animació de la simulació. A la pantalla "Options > Animation Options". En el primer camp podrem definir el temps d'espera entre els dispars de dues transicions. Mentre que el segon camp serveix per definir el temps durant el qual cada component afectat pel dispar d'una transició es ressalti.



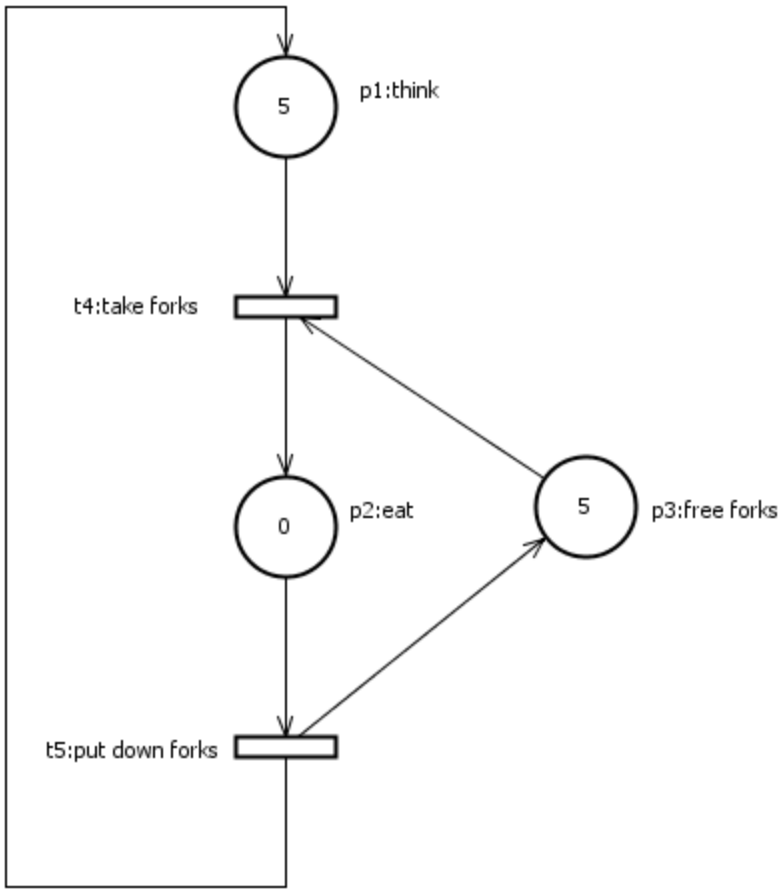
7.6. Xarxa de Petri Acolorida: Dinar de Filòsofs

En aquest nou exemple introduïm el concepte de color. En el nostre cas un color serà representat per un objecte Java.

Es modelarà una xarxa de Petri que simularà el conegut problema del dinar de filòsofs.

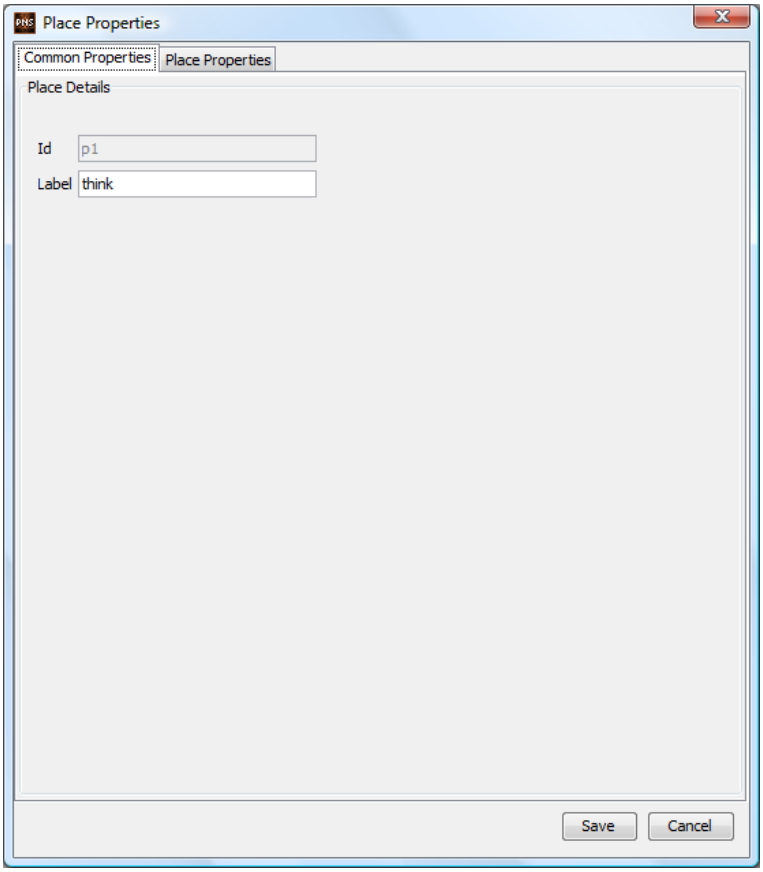
7.6.1. Disseny del model

Primer cal dissenyar un model com el de la següent figura.



Per modificar les etiquetes de cada component de la xarxa, caldrà utilitzar l'eina de selecció ( ) i fer un sol clic amb el botó dret del ratolí sobre el component.

S’obrirà una finestra com aquesta.

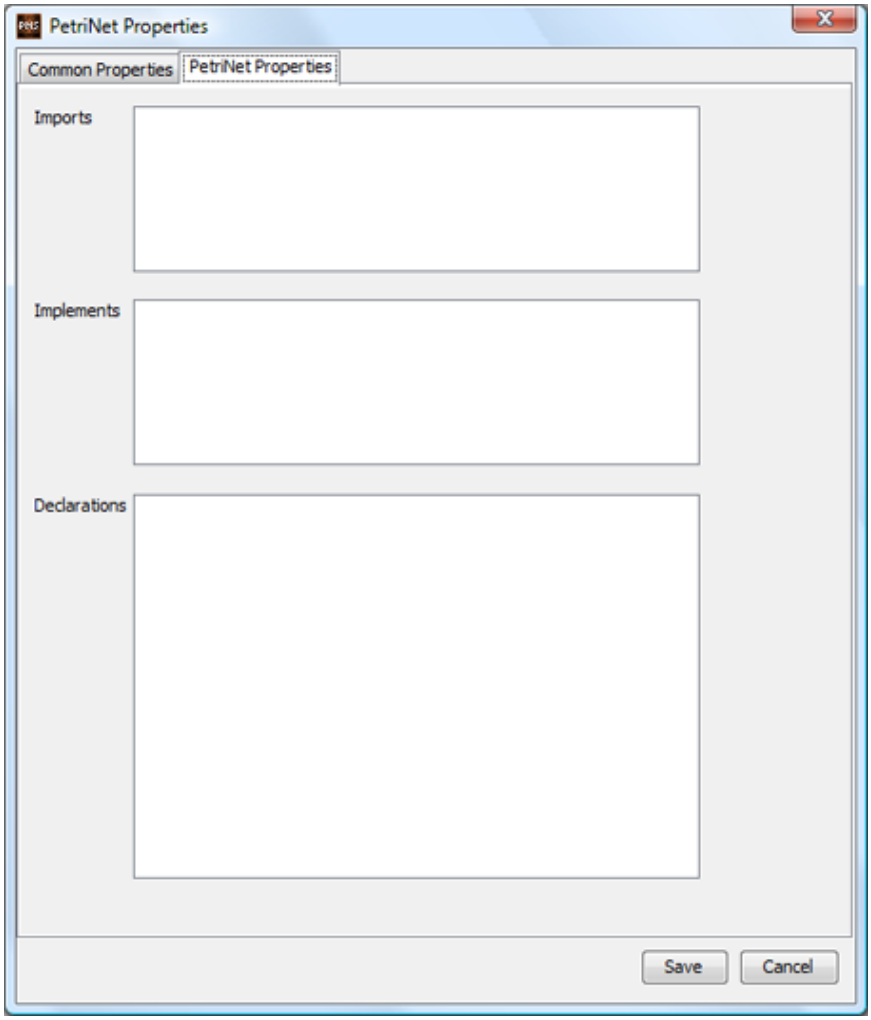


La primera pestanya és comuna a tots els objectes de la xarxa, mentre que la següent pestanya és pròpia de cada element.

Es recomana utilitzar els mateixos noms a les etiquetes que a l’exemple per seguir correctament les següents indicacions.

7.6.2. Propietats de la xarxa

Com que al simular una xarxa de Petri amb PetriNetSim el que realment es fa és compilar i executar una classe Java, des de l'editor també es permet afegir codi personalitzat que després completarà aquesta classe. Les declaracions, la implementació d'altres classes i les importacions de llibreries es poden afegir al model si es requereix. Només cal fer clic amb el botó dret sobre qualsevol punt de l'àrea de disseny on no hi hagi cap objecte. S'obrirà una finestra com la de la següent figura on, a part de poder modificar l'etiqueta que representarà el nom de la classe, es podrà introduir el codi propi.



A imports afegirem la instrucció

```
import java.util.Iterator;
```

En aquest cas no implementarem cap classe, i deixarem aquest camp buit.

En el camp Declaracions afegirem el següent codi:

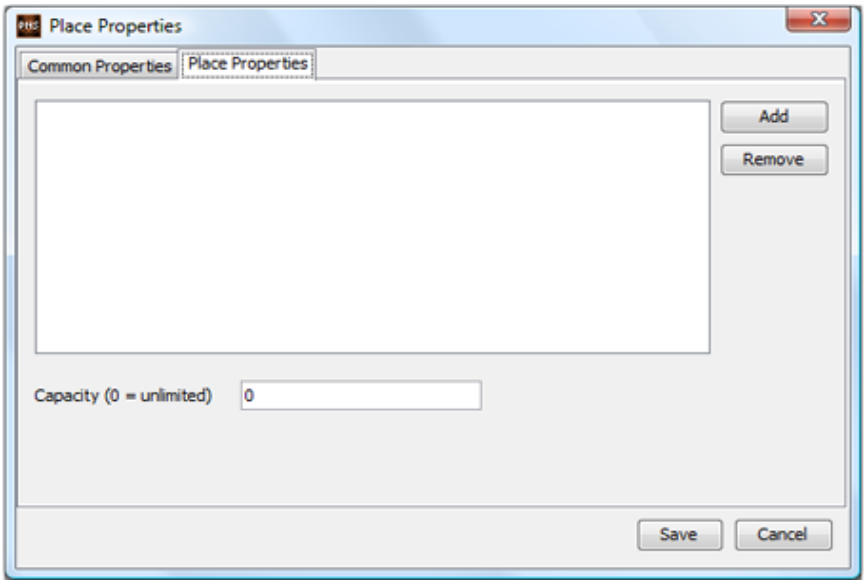
```
int x = 1;

    public String left(int i) {
        String s = "";
        switch (i) {
            case 1:
                s = new String("b");
                break;
            case 2:
                s = new String("c");
                break;
            case 3:
                s = new String("d");
                break;
            case 4:
                s = new String("e");
                break;
            case 5:
                s = new String("f");
                break;
        }
        return s;
    }

    public String right(int i) {
        String s = "";
        switch (i) {
            case 1:
                s = new String("a");
                break;
            case 2:
                s = new String("b");
                break;
            case 3:
                s = new String("c");
                break;
            case 4:
                s = new String("d");
                break;
            case 5:
                s = new String("e");
                break;
        }
        return s;
    }

    public TokenSet lr(int i) {
        TokenSet tk = new TokenSet(new Token(right(i)));
        tk.add(new Token(left(i)));
        return tk;
    }
```

El que fa aquest codi és inicialitzar en una primera instància la variable x que representa a un filòsof, després la funció **left()** retorna la forquilla de l'esquerra del

filòsof i la funció **right()** la de la dreta. La funció **lr()** servirà per selecciona la forquilla esquerra i dreta del filòsof.

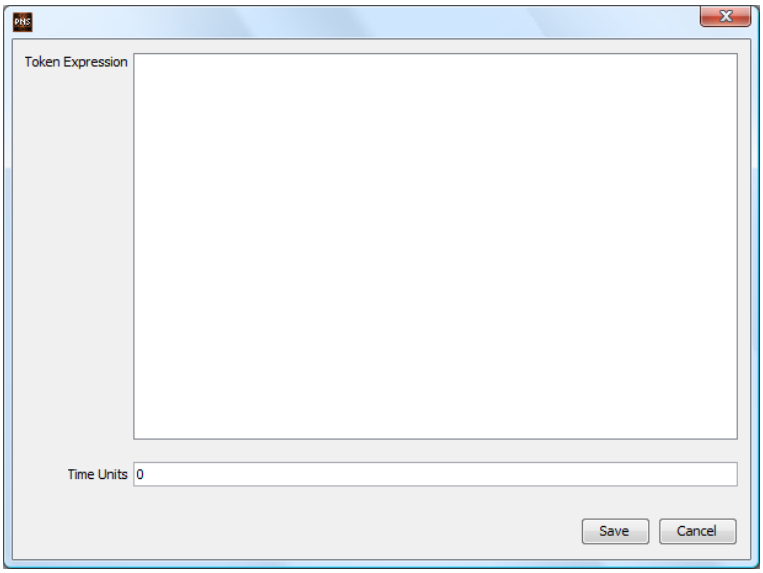
7.6.3. Afegir codi als altres components

Ara seleccionarem amb el botó dret sobre el lloc "think" per introduir-hi 5 fitxes inicials. Això ho farem des de la pestanya "Place Properties".



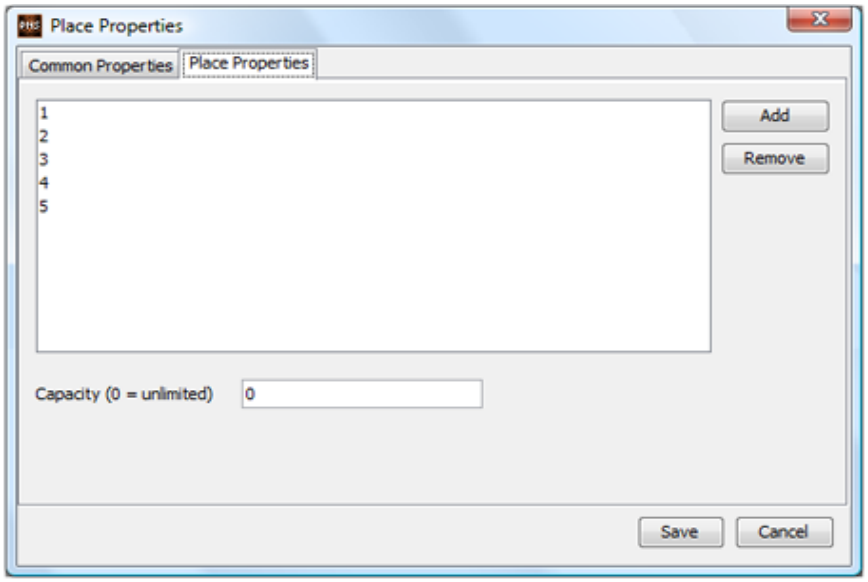
El camp Capacitat permet afegir una restricció de capacitat al número de fitxes que hi poden cabre a un lloc. Si no es vol que hi hagi la restricció es pot deixar a 0.

Es prem sobre "Add" i s'obrirà una finestra on podem escriure una expressió en Java.



En aquest exemple encara no utilitzarem el temps, de manera que el camp “Time Units” el podem deixar a 0.

En aquest lloc hi entrarem els 5 filòsofs que els representarem amb enters, de forma que la “Token Expression” que representaria al primer filòsof seria el nombre enter 1. El segon filòsof seria el 2 i així successivament. Quedaria d’aquesta forma.



No s’ha d’afegir punt i coma per finalitzar l’expressió al crear una nova fitxa.

A continuació caldrà modificar la funció de guarda de la transició “take forks”. A la pestanya de propietats de la transició entrem aquest codi:

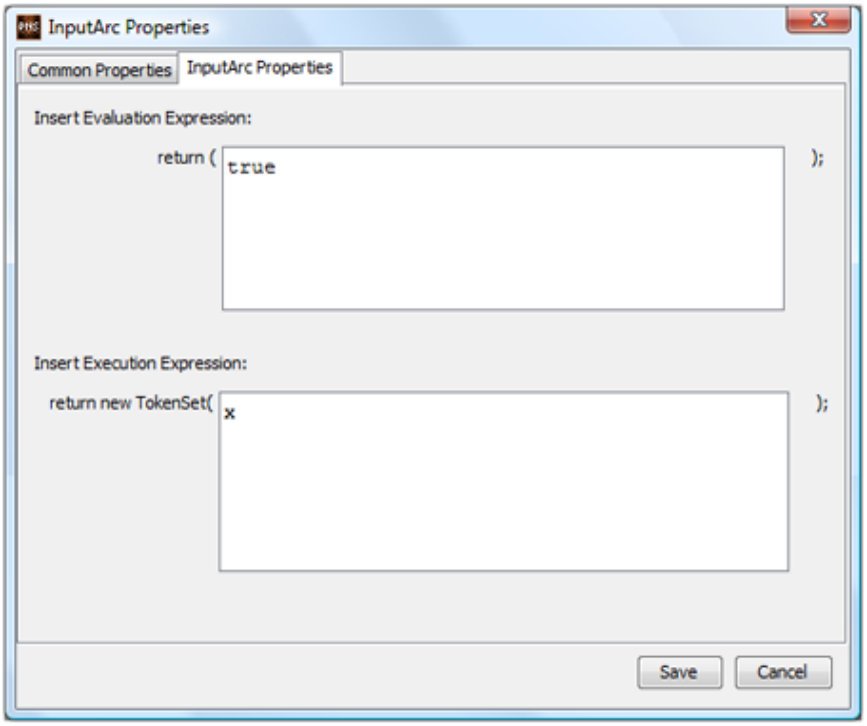
```
boolean found = false;
Iterator it = p1.getTokens().iterator();
while (!found && it.hasNext()) {
    Token token = (Token) it.next();
    int i = (Integer) token.getObject();
    if (p3.getTokens().containsAll(lr(i))) {
        x = i;
        found = true;
    }
}
return found;
```

El que fa aquest codi és comprovar si hi ha algun filòsof que tingui disponibles la forquilla de la mà esquerra i de la dreta, si n’hi ha algun es complirà la funció de guarda i la transició s’executarà. Cal destacar que és aquí on la variable x pren el valor del filòsof. Aquest valor serà d’utilitat més endavant.

Possiblement s’hagi de modificar les referències del codi als llocs ja que poden tenir altres identificadors. En el cas de l’exemple, el lloc **p1** correspon al lloc amb etiqueta “think” i el lloc **p3** correspon al lloc “free forks”.

Les expressions de guarda de les transicions cal acabar-les amb punt i coma. Les expressions dels arcs no.

Seguidament modificarem l’arc d’entrada que va del lloc “think” fins a la transició “free forks”.



D'expressió d'avaluació no en necessitem cap d'especial per el que ho deixem en "true". Mentre que a l'expressió d'execució afegim la variable x per tal que esborri del lloc una fitxa amb el valor d'aquesta variable.

A l'arc de sortida que va de la transició "take forks" a eat a l'expressió d'execució li passarem també la variable x.

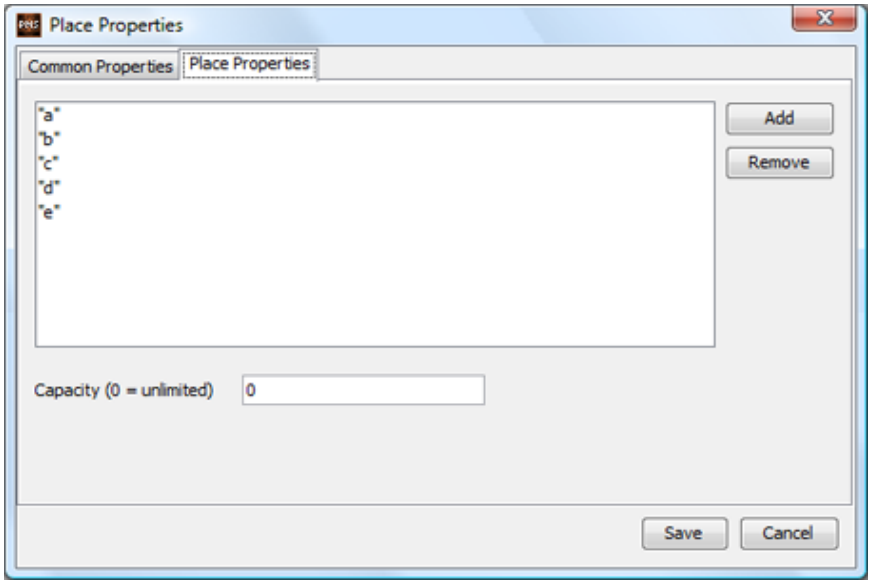
A l'expressió d'execució de l'arc que va del lloc "eat" a la transició "put down forks" escriurem l'expressió:

```
x = (Integer) (getTokenSet().get(0)).getObject()
```

Aquesta expressió assigna el valor del primer objecte que trobi del lloc a la variable x. Recordem que les fitxes es guarden dins dels llocs segons van arribant seguint la premissa "First In First Out", per això utilitzant l'índex 0 ens assegurem d'agafar el primer element que havia entrat.

Ara cal afegir les fitxes que representen les forquilles. Donat que les forquilles són un altre "color" cal distingir-les dels filòsofs assignant-l'hi un altre tipus. Podem designar-les per exemple com a tipus cadena ("String"). Les cadenes s'han d'introduir entre

cometes. La primera forquilla es representarà per la lletra “a” fins a la cinquena per la “e”.



Finalment per acabar assignarem la funció

```
lr(x)
```

com a expressió d’execució tant a l’arc que va de “put down forks” a “free forks” com a l’arc que va de “free forks” a “take forks” i la variable

```
x
```

a l’arc que va de “put down forks” a “think”.

Ara ja es pot executar el model i veurem que a diferencia del mateix model del dinar de filòsofs amb xarxes de Petri normals no hi ha deadlocks gràcies a la possibilitat de poder escollir que ens ofereix la programació.

Hi ha la possibilitat de veure el codi que genera l’editor i que serà la classe que es compilarà internament per a fer la simulació des de la icona de la barra d’eines ().

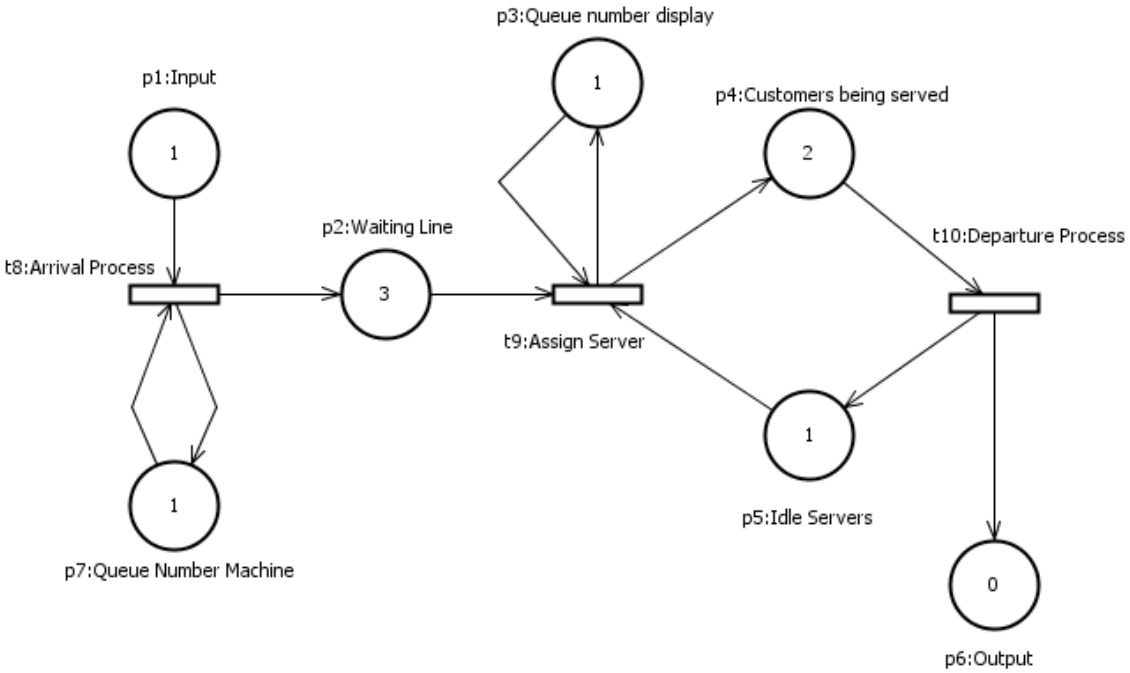
7.7. Xarxa de Petri Acolorida Temporal: Aeroport

A continuació modelarem una nova xarxa de Petri per veure en funcionament el concepte de temps.

En aquest exemple modelarem una possible situació que podria ocórrer en una terminal d'un aeroport, o a l'embarcament d'un port on el usuari ha de validar el bitllet per poder embarcar. En el sistema bàsicament hi ha gent que arriba a l'aeroport, i s'espera en una cua fins que se'l pot atendre i finalment embarcar.

7.7.1. Disseny del model

El disseny gràfic del model és així:



7.7.2. Inicialitzacions i declaracions

Primer de tot cal definir les importacions d'altres classes i les declaracions a les propietats de la xarxa.

Com a importacions afegim:

```
import java.util.Iterator;
```

Com a declaracions afegim:

```
String customer;
int queueNumber;
String server;
class CheckIn{
  private String customer;
  private int queueNumber;
  public CheckIn(String customer, int queueNumber){
    this.customer = customer;
    this.queueNumber = queueNumber;
  }
  public String toString(){
    return "Customer: "+this.customer+" | Q.Number:
"+this.queueNumber;
  }
  public int getQueueNumber(){
    return this.queueNumber;
  }
  public String getCustomer(){
    return this.customer;
  }
}
class  UseServer{
  private String customer;
  private String server;
  public UseServer(String customer, String server){
    this.customer = customer;
    this.server = server;
  }
  public String toString(){
    return "Customer: "+this.customer+" | Server: "+this.server;
  }
  public String getServer(){
    return this.server;
  }
  public String getCustomer(){
    return this.customer;
  }
}

public CheckIn removeCheckIn(CheckIn checkIn){
    this.customer = checkIn.getCustomer();
  return checkIn;
}

public Token removeUseServer(InputArc inputArc) {
    Token token =
inputArc.removeTimedToken(inputArc.getPlace().getTokens());
    UseServer u = (UseServer) (token.getObject());
    this.customer = u.getCustomer();
    this.server = u.getServer();
```

```
    return token;
}
```

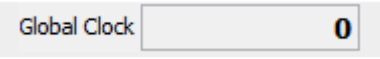
En el codi definim un client “customer” que serà la persona que anirà passant pels diferents estats del sistema, un número de cua “queuenumber” que indicarà en cadascuna de les cues quin número té el client en aquell moment i un servidor “server” que indicarà el servidor que està usant un client en un determinat moment.

Hi ha dues classes “CheckIn” i “UseServer”. La primera servirà per assignar un número de cua a un client i la segona per assignar un servidor a un client.

Com en el cas de les classes “CheckIn” i “UseServer” de l’exemple, és recomanable sobreescriure el mètode **toString()** per representar l’objecte de forma llegible al disseny.

Finalment les funcions removeCheckIn() i removeUseServer(). La primera serveix per especificar quina fitxa s’eliminarà i de pas assignar el valor d’aquesta fitxa que representa un client a la variable “customer”. El segon mètode serveix per eliminar l’assignació client-servidor i de pas també, assignar aquests valors a les respectives variables.

La funció **removeTimedToken()** es la que es farà servir sempre que es vulgui eliminar fitxes amb marques de temps d’un lloc. Com a paràmetre se li ha de passar el conjunt de fitxes sobre les que es vulgui avaluar i on es troba alguna amb una marca de temps menor o igual al temps global de la simulació. Aquest temps global el podem veure indicat a la dreta de la barra d’eines. Marca les unitats de temps de la simulació global i és el temps que s’anirà incrementant sempre que no hi hagi fitxes per ser disparades.



7.7.3. Funcions de guarda i creació de fitxes

Començarem per afegir una fitxa al lloc "Input". Aquest lloc representa les persones que arriben a l'aeroport. En aquest cas per l'exemple afegirem un nou client representat per una cadena de text

```
"Jack"
```

Recordem que les cadenes de text s'han d'afegir entre cometes.

A l'arc d'entrada entre el lloc "Input" i la transició "Arrival Process" com a expressió d'execució afegim:

```
customer = (String) (getTokenSet().get(0)).getObject()
```

Al lloc "Queue Number Machine" afegim una fitxa de valor

```
7
```

que representa l'últim número de cua assignat

A l'arc de sortida que va entre la transició "Arrival Process" i el lloc "Queue Number Machine" afegirem l'expressió d'execució:

```
queueNumber + 1
```

que incrementarà en una unitat el número de tanda.

I en l'arc d'entrada entre el lloc "Queue Number Machine" i la transició "Arrival Process" afegirem:

```
queueNumber = (Integer) (getTokenSet().get(0)).getObject()
```

que servirà per assignar el número de tanda que toca al crear fitxes a l'arc de sortida que va de la transició "Arrival Process" a "Waiting Line". En aquest arc cal afegir l'expressió:

```
new CheckIn(customer,queueNumber)
```

on s'assigna el client amb aquest número de cua.

Al lloc "Waiting Line" s'ha afegit 3 fitxes que representen l'assignació client-número de cua que, a tall d'exemple, afegim un a un al lloc:

```
new CheckIn("Marta",4)
new CheckIn("Silvia",5)
new CheckIn("Jessica",6)
```

A l'arc d'entrada que va del lloc "Waiting Line" a la transició "Assign Server" afegim l'expressió d'execució:

```
removeCheckIn((CheckIn) (getTokenSet().get(0)).getObject())
```

Ara repetim el procés que hem fet abans per a crear una cua. Al lloc "Queue number Display" afegim una fitxa de valor

```
4
```

que representa l'últim número de cua assignat.

A l'arc d'entrada entre que va des del lloc "Queue number display" i la transició "Assign Server" afegim l'expressió d'execució:

```
queueNumber = (Integer) ((Token)
getTokenSet().get(0)).getObject()
```

I a l'arc de sortida entre els dos mateixos nodes

```
queueNumber + 1
```

En l'arc de sortida entre la transició "Assign Server" i el lloc "Customers being served" afegirem l'expressió

```
new UseServer(customer,server)
```

que representa l'assignació d'un client a un servidor.

I en aquest mateix arc cal afegir l'expressió de temps, que serà el temps afegit a cada fitxa quan es creï al disparar-se la transició. S'afegeix al camp "Time Units":

```
60
```

Com a expressió de temps també es podria introduir la crida a una funció de temps, només és necessari que retorni un valor de tipus **long**.

Al lloc "Customers being served" afegim unes quantes fitxes a mode d'exemple:

```
Expressió:
new UseServer("Laia","Calipso")
Time Units: 324
Expressió:
new UseServer("Lorena","Cassiopeia")
Time Units: 349
```

A l'arc d'entrada que va del lloc "Customers being served" a la transició "Departure Process" cal afegir l'expressió:

```
removeUseServer(this)
```

A l'arc que va de la transició "Departure Process" al lloc "Output" afegirem com a expressió la variable

```
customer
```

Mentre que a l'arc que va de la transició "Departure Process" al lloc "Idle Servers" afegirem la variable

```
server
```

Els servidors els representarem mitjançant cadenes de text. A mode d'exemple hem afegit un servidor al lloc "Idle Servers" anomenat:

```
"Server 1: Andromeda"
```

Finalment, a l'arc d'entrada que va del lloc "Idle Servers" a la transició "Assign Server" cal afegir com a expressió d'execució :

```
server = (String) (getTokenSet().get(0)).getObject()
```

per assignar a la variable server el primer servidor disponible que h hagi al lloc "Idle Servers".

7.8. Resum de mètodes útils

Per a cada component de la xarxa de Petri hi ha un seguit de funcions i mètodes que ajudaran a manipular dades del model. Tot i que les més importants ja han aparegut en els exemples anteriors, seguidament s'expliquen detalladament una per una:

7.8.1. Lloc

getTokens():TokenSet. Mètode que retorna el conjunt de fitxes que hi ha actualment en el lloc.

setTokens(TokenSet):void. Mètode que permet assignar un nou conjunt de fitxes al lloc actual. Descarta les fitxes que ja hi ha, si n'hi ha.

addToken(TokenSet):void. Permet afegir un conjunt de fitxes al lloc a continuació de les que ja hi ha.

removeTokens(TokenSet):void. Permet eliminar un conjunt de fitxes del lloc actual.

getCapacity():int. Retorna la capacitat màxima del lloc, és a dir, el nombre màxim de fitxes que pot contenir.

setCapacity():void. Estableix la capacitat màxima del lloc.

7.8.2. Transició

getGlobalClock():long. Retorna el valor del rellotge global de la simulació.

7.8.3. Arc d'entrada "InputArc"

getPlace():Place. Retorna el lloc a que fa referència l'arc.

getTransition():Transition. Retorna la transició a que fa referència d'arc.

getTokenSet():TokenSet. Retorna les fitxes que conté el lloc. És equivalent a fer getPlace.getTokenSet()

removeTimedToken(TokenSet):Token. Donat un conjunt de fitxes retorna una fitxa amb marca de temps menor o igual al rellotge de simulació.

7.8.4. Arc de Sortida "OutputArc"

getPlace():Place. Retorna el lloc a que fa referència l'arc.

getTransition():Transition. Retorna la transició a que fa referència d'arc.

getTokenSet():TokenSet. Retorna les fitxes que conté el lloc. És equivalent a fer getPlace.getTokenSet()

7.8.5. TokenSet

Recordem que un TokenSet és una extensió de la classe AbstractCollection de Java, per tant ofereix la possibilitat de fer servir mètodes d'aquesta classe.

isEmpty():boolean. Retorna cert si la col·lecció és buida, fals altrament.

size():int. Retorna el nombre d'elements que conté la col·lecció.

clear():void. Elimina tots els elements de la col·lecció.

get(int):Token. Retorna la fitxa que hi ha a la posició passada per paràmetre del conjunt de fitxes.

7.8.6. Token

equals(objecte):boolean. Comprova si l'objecte que se li passa per paràmetre és igual que la fitxa.

toString():String. Serveix per obtenir una representació en forma de cadena de l'objecte. És útil per visualitzar la fitxa durant la simulació.

getObject():Object. Retorna l'objecte que representa la fitxa, el "color".

setObject(objecte):void. Assigna un nou objecte a la fitxa.

getTimestamp():long. Retorna el valor de la marca de temps de la fitxa.

setTimestamp(long):void. Assigna una nova marca de temps a la fitxa.

Per a més informació es pot consultar el document JavaDoc on es mostra informació de totes les classes i mètodes del projecte.

8. Proves

Per validar l'aplicació s'han fet un seguit de proves que han consistit en l'execució de diferents exemples de xarxes de Petri. Aquests exemples es poden trobar al CD adjunt al projecte. S'ha provat que el comportament lògic de la simulació del model fos el correcte. I també que l'ús de les eines del programa per a crear i manipular els models fos l'adequat.

A més s'ha rebut l'ajuda d'un alumne que també ha col·laborat en la validació de l'aplicació, tot determinant errades del programa, així com suggeriments de com millorar alguna funcionalitat.

9. Possibles ampliacions

Sempre hi ha possibles millores o noves funcionalitats a afegir a una aplicació. A continuació es plantegen algunes idees que es podrien implementar per fer de PetriNetSim un programa més complet.

9.1. Implementar noves extensions

Implementar arcs inhibidors i arcs de test.

Afegir suport de temps continu en la simulació de xarxes de Petri temporals.

Introduir eines d'anàlisi de xarxes de Petri.

9.2. Afegir funcionalitats a l'editor

Crear una eina per copiar i enganxar objectes de l'àrea de dibuix.

Afegir la possibilitat d'ampliar o disminuir l'àrea de dibuix.

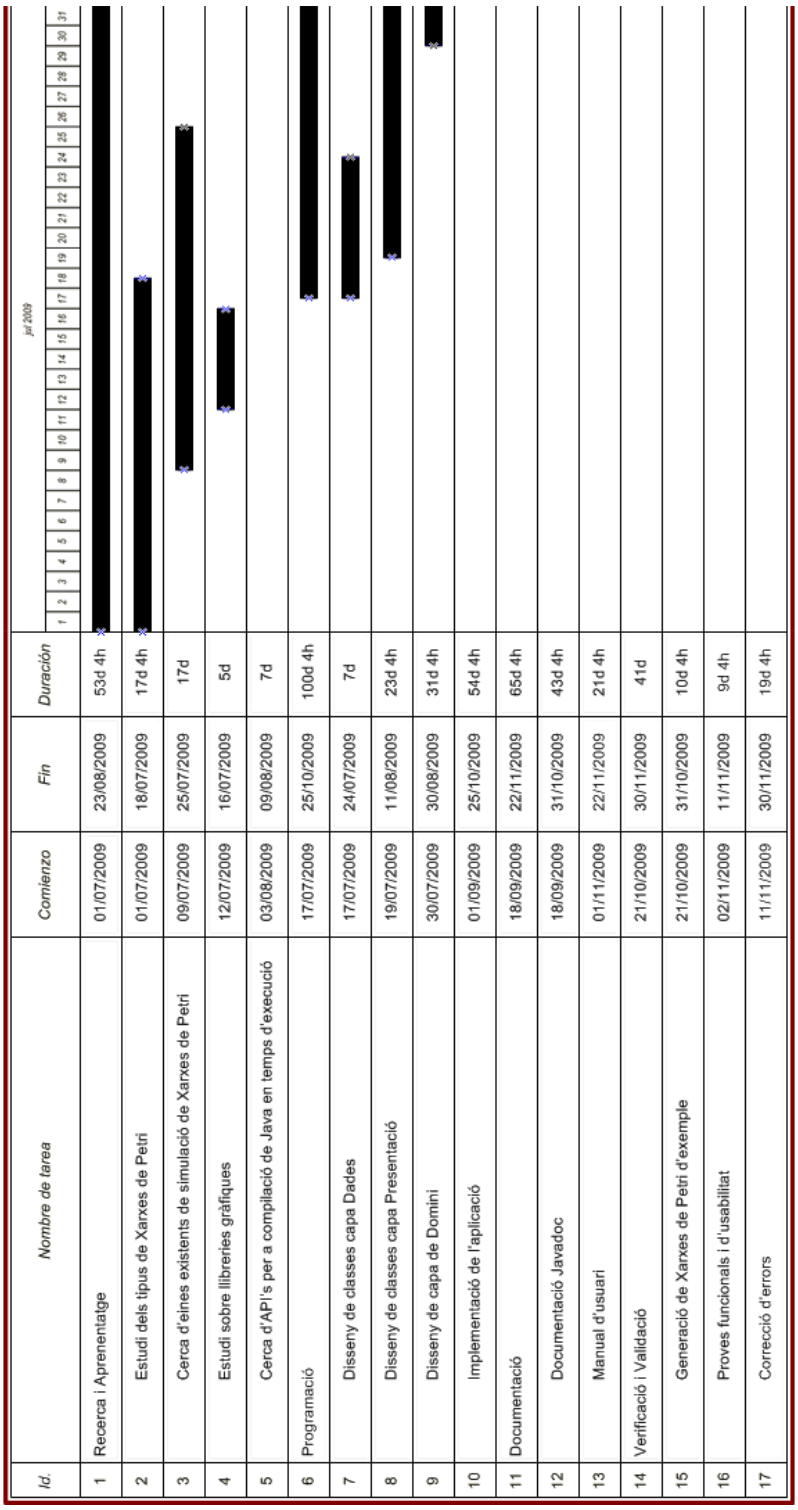
Augmentar o disminuir l'àrea de dibuix automàticament quan es creïn models grans.

Dissenyar un sistema per desfer canvis.

Permetre obrir més d'una xarxa de Petri simultàniament.

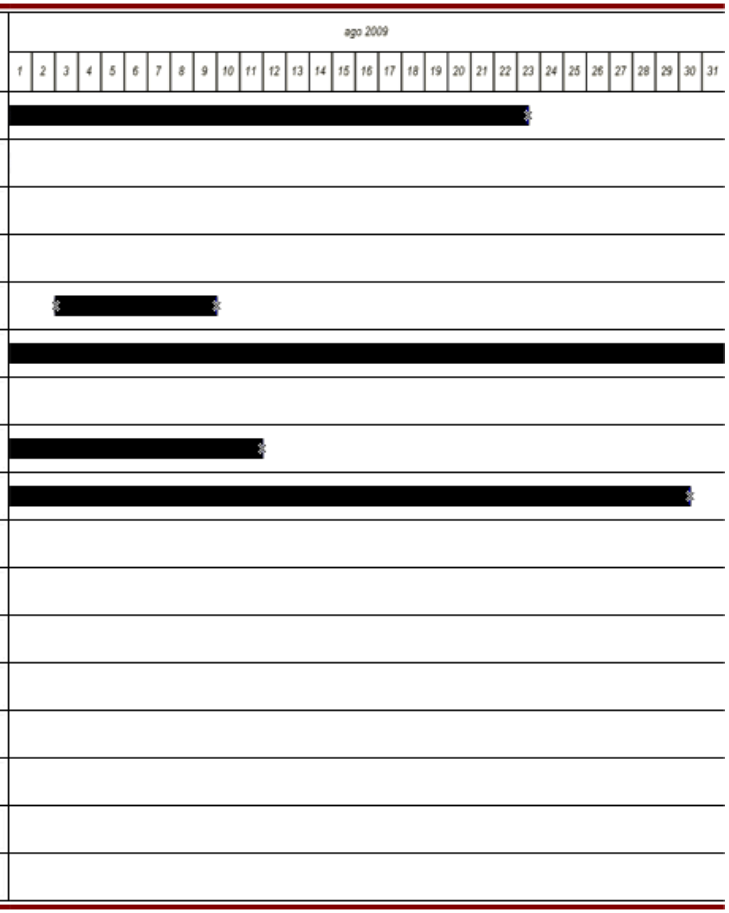
10. Planificació

A continuació es mostra la planificació plantejada per a la realització d'aquest projecte.

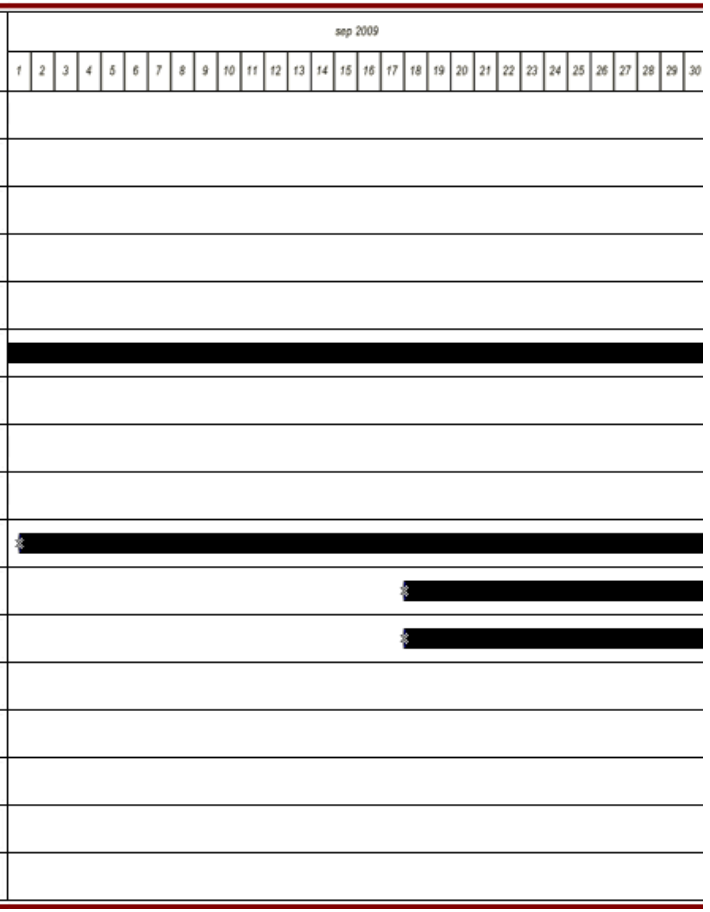


Id.	Nombre de tarea	Comienzo	Fin	Duración
1	Recerca i Aprenentatge	01/07/2009	23/08/2009	53d 4h
2	Estudi dels tipus de Xarxes de Petri	01/07/2009	18/07/2009	17d 4h
3	Cerca d'eines existents de simulació de Xarxes de Petri	09/07/2009	25/07/2009	17d
4	Estudi sobre llibreries gràfiques	12/07/2009	16/07/2009	5d
5	Cerca d'API's per a compilació de Java en temps d'execució	03/08/2009	09/08/2009	7d
6	Programació	17/07/2009	25/10/2009	100d 4h
7	Disseny de classes capa Dades	17/07/2009	24/07/2009	7d
8	Disseny de classes capa Presentació	19/07/2009	11/08/2009	23d 4h
9	Disseny de capa de Domini	30/07/2009	30/08/2009	31d 4h
10	Implementació de l'aplicació	01/09/2009	25/10/2009	54d 4h
11	Documentació	18/09/2009	22/11/2009	65d 4h
12	Documentació Javadoc	18/09/2009	31/10/2009	43d 4h
13	Manual d'usuari	01/11/2009	22/11/2009	21d 4h
14	Verificació i Validació	21/10/2009	30/11/2009	41d
15	Generació de Xarxes de Petri d'exemple	21/10/2009	31/10/2009	10d 4h
16	Proves funcionals i d'usabilitat	02/11/2009	11/11/2009	9d 4h
17	Correcció d'errors	11/11/2009	30/11/2009	19d 4h

Id.	Nombre de tarea	Comienzo	Fin	Duración
1	Recerca i Aprenentatge	01/07/2009	23/08/2009	53d 4h
2	Estudi dels tipus de Xarxes de Petri	01/07/2009	18/07/2009	17d 4h
3	Cerca d'eines existents de simulació de Xarxes de Petri	09/07/2009	25/07/2009	17d
4	Estudi sobre llibreries gràfiques	12/07/2009	16/07/2009	5d
5	Cerca d'API's per a compilació de Java en temps d'execució	03/08/2009	09/08/2009	7d
6	Programació	17/07/2009	25/10/2009	100d 4h
7	Disseny de classes capa Dades	17/07/2009	24/07/2009	7d
8	Disseny de classes capa Presentació	19/07/2009	11/08/2009	23d 4h
9	Disseny de capa de Domini	30/07/2009	30/08/2009	31d 4h
10	Implementació de l'aplicació	01/09/2009	25/10/2009	54d 4h
11	Documentació	18/09/2009	22/11/2009	65d 4h
12	Documentació Javadoc	18/09/2009	31/10/2009	43d 4h
13	Manual d'usuari	01/11/2009	22/11/2009	21d 4h
14	Verificació i Validació	21/10/2009	30/11/2009	41d
15	Generació de Xarxes de Petri d'exemple	21/10/2009	31/10/2009	10d 4h
16	Proves funcionals i d'usabilitat	02/11/2009	11/11/2009	9d 4h
17	Correcció d'errors	11/11/2009	30/11/2009	19d 4h

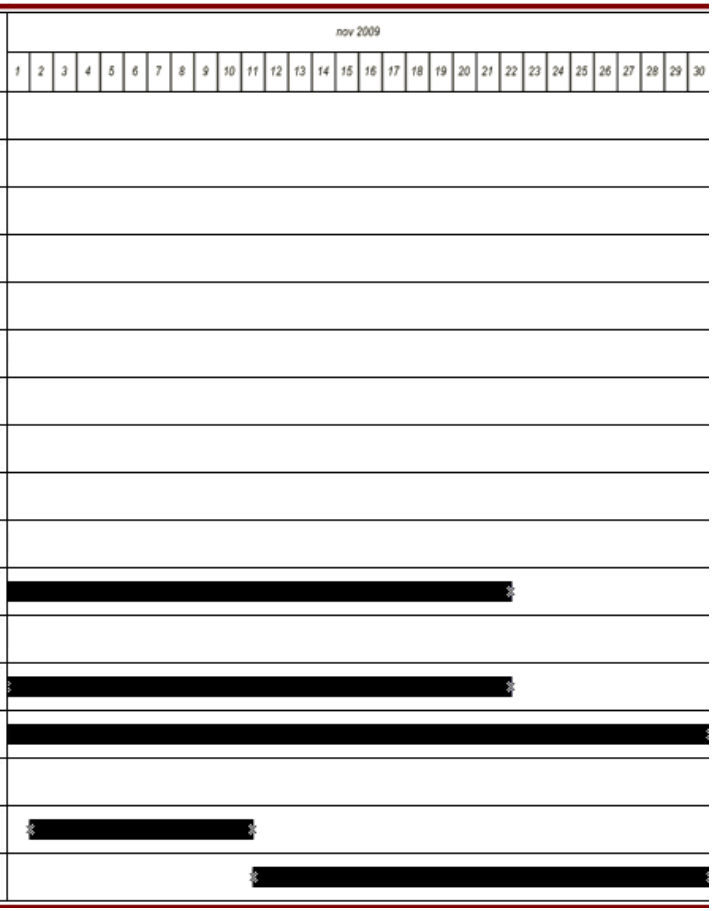


Id.	Nombre de tarea	Comienzo	Fin	Duración
1	Recerca i Aprenentatge	01/07/2009	23/08/2009	53d 4h
2	Estudi dels tipus de Xarxes de Petri	01/07/2009	18/07/2009	17d 4h
3	Cerca d'eines existents de simulació de Xarxes de Petri	09/07/2009	25/07/2009	17d
4	Estudi sobre llibreries gràfiques	12/07/2009	16/07/2009	5d
5	Cerca d'API's per a compilació de Java en temps d'execució	03/08/2009	09/08/2009	7d
6	Programació	17/07/2009	25/10/2009	100d 4h
7	Disseny de classes capa Dades	17/07/2009	24/07/2009	7d
8	Disseny de classes capa Presentació	19/07/2009	11/08/2009	23d 4h
9	Disseny de capa de Domini	30/07/2009	30/08/2009	31d 4h
10	Implementació de l'aplicació	01/09/2009	25/10/2009	54d 4h
11	Documentació	18/09/2009	22/11/2009	65d 4h
12	Documentació Javadoc	18/09/2009	31/10/2009	43d 4h
13	Manual d'usuari	01/11/2009	22/11/2009	21d 4h
14	Verificació i Validació	21/10/2009	30/11/2009	41d
15	Generació de Xarxes de Petri d'exemple	21/10/2009	31/10/2009	10d 4h
16	Proves funcionals i d'usabilitat	02/11/2009	11/11/2009	9d 4h
17	Correcció d'errors	11/11/2009	30/11/2009	19d 4h



Id.	Nombre de tarea	Comienzo	Fin	Duración
1	Recerca i Aprenentatge	01/07/2009	23/08/2009	53d 4h
2	Estudi dels tipus de Xarxes de Petri	01/07/2009	18/07/2009	17d 4h
3	Cerca d'eines existents de simulació de Xarxes de Petri	09/07/2009	25/07/2009	17d
4	Estudi sobre llibreries gràfiques	12/07/2009	16/07/2009	5d
5	Cerca d'API's per a compilació de Java en temps d'execució	03/08/2009	09/08/2009	7d
6	Programació	17/07/2009	25/10/2009	100d 4h
7	Disseny de classes capa Dades	17/07/2009	24/07/2009	7d
8	Disseny de classes capa Presentació	19/07/2009	11/08/2009	23d 4h
9	Disseny de capa de Domini	30/07/2009	30/08/2009	31d 4h
10	Implementació de l'aplicació	01/09/2009	25/10/2009	54d 4h
11	Documentació	18/09/2009	22/11/2009	65d 4h
12	Documentació Javadoc	18/09/2009	31/10/2009	43d 4h
13	Manual d'usuari	01/11/2009	22/11/2009	21d 4h
14	Verificació i Validació	21/10/2009	30/11/2009	41d
15	Generació de Xarxes de Petri d'exemple	21/10/2009	31/10/2009	10d 4h
16	Proves funcionals i d'usabilitat	02/11/2009	11/11/2009	9d 4h
17	Correcció d'errors	11/11/2009	30/11/2009	19d 4h

Id.	Nombre de tarea	Comienzo	Fin	Duración
1	Recerca i Aprenentatge	01/07/2009	23/08/2009	53d 4h
2	Estudi dels tipus de Xarxes de Petri	01/07/2009	18/07/2009	17d 4h
3	Cerca d'eines existents de simulació de Xarxes de Petri	09/07/2009	25/07/2009	17d
4	Estudi sobre llibreries gràfiques	12/07/2009	16/07/2009	5d
5	Cerca d'API's per a compilació de Java en temps d'execució	03/08/2009	09/08/2009	7d
6	Programació	17/07/2009	25/10/2009	100d 4h
7	Disseny de classes capa Dades	17/07/2009	24/07/2009	7d
8	Disseny de classes capa Presentació	19/07/2009	11/08/2009	23d 4h
9	Disseny de capa de Domini	30/07/2009	30/08/2009	31d 4h
10	Implementació de l'aplicació	01/09/2009	25/10/2009	54d 4h
11	Documentació	18/09/2009	22/11/2009	65d 4h
12	Documentació Javadoc	18/09/2009	31/10/2009	43d 4h
13	Manual d'usuari	01/11/2009	22/11/2009	21d 4h
14	Verificació i Validació	21/10/2009	30/11/2009	41d
15	Generació de Xarxes de Petri d'exemple	21/10/2009	31/10/2009	10d 4h
16	Proves funcionals i d'usabilitat	02/11/2009	11/11/2009	9d 4h
17	Correcció d'errors	11/11/2009	30/11/2009	19d 4h



11. Valoracions econòmiques

Aquest projecte es pot dividir en vàries tasques com són:

Recerca i Aprenentatge. 90 h. Que inclou l'estudi de xarxes de Petri, models de representació XML i tot el que ha estat necessari aprendre per poder implementar l'aplicació en Java (estudi de la llibreria Swing, diferents estratègies per programar funcionalitats, etc).

Programació. 180 h. Comprèn des del disseny de les classes fins a la implementació del codi.

Documentació. 158 h. Consta del manual i guia d'usuari així com de la documentació del propi codi plasmat en un document HTML JavaDoc i en l'explicació de com s'ha implementat cada funcionalitat.

Verificació i Validació. 22 h. És la realització de proves per constatar que l'editor es comporta tal i com s'havia dissenyat i també serveix per corregir possibles errades realitzades durant la programació.

Finalment, és necessari un anàlisi del cost

PROGRAMADOR	Cost (€/h)	Hores	Total (€)
Guillem Català	20	450	9000

El preu total d'aquest projecte és de 9000€.

12. Continguts CD

Aquest projecte adjunta un CD que conté:

- El programa PetriNetSim.
- El projecte NetBeans de PetriNetSim que inclou el codi font del mateix.
- El document JavaDoc de la documentació de les classes.
- Una còpia d'aquesta memòria en format .docx.
- Una còpia d'aquesta memòria en format .pdf.

13. Bibliografia

13.1. Sobre xarxes de Petri

[1] http://www.daimi.au.dk/CPnets/intro/
Informació sobre xarxes de Petri Acolorides.

[2] http://www.informatik.uni-hamburg.de/TGI/PetriNets/classification/level3/CPN.html
Definició i exemple de xarxa de Petri acolorida.

[3] http://www.cs.unc.edu/~montek/teaching/spring-04/petrinets.ppt
Altres tipus de xarxes de Petri.

[4] http://www.informatik.uni-hamburg.de/TGI/PetriNets/
Web dedicada al món de les xarxes de Petri, amb informació teòrica, editors i simuladors de xarxes de Petri, informació de recerca, etc.

13.2. Per la creació del projecte

[5] http://netbeans.org/
Entorn de desenvolupament usat per implementar el codi Java de l'aplicació i generar diagrames UML.

[6] http://office.microsoft.com/es-es/visio/FX100487863082.aspx
Eina per crear diagrames de Gantt.